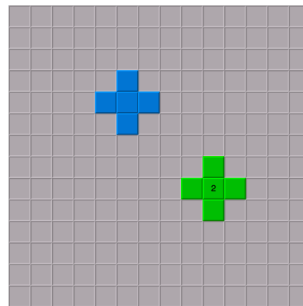


# The Liquid Metal Blokus Experiment



Stephen Fink

IBM Research

## Liquid Metal Team (IBM T. J. Watson Research Center)



Josh Auerbach



David Bacon



Ioana Baldini



Perry Cheng



Stephen Fink



Rodric Rabbah



Sunil Shukla

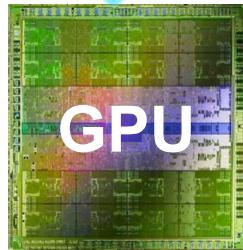
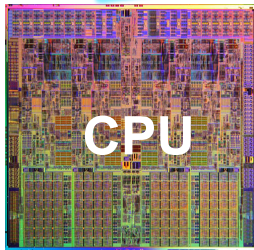
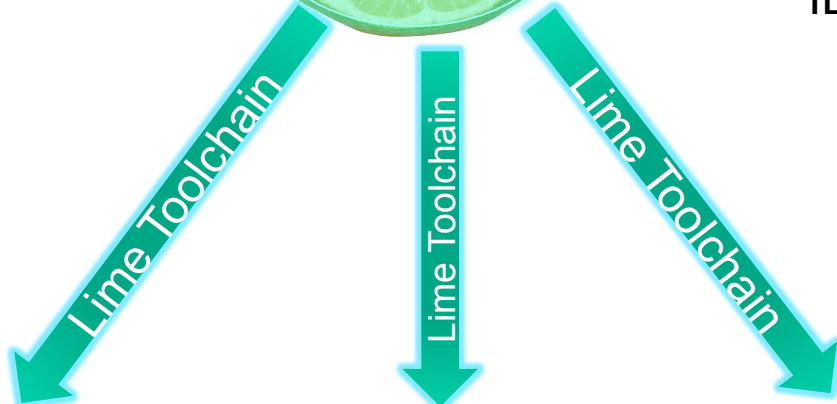
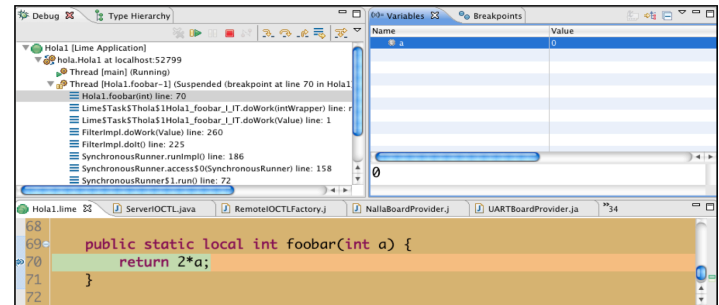
# Goal: Make Heterogeneous Platforms Accessible for Mainstream Use

## Vision:

## High-Level Java-Derived Language

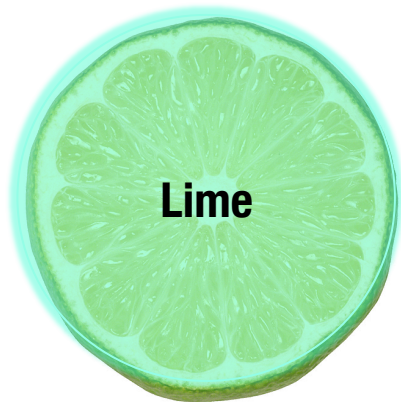
```
public static local int foobar(int a) {
    return 2*a;
}
```

## IDE, Compiler, Debugger, Cloud Synthesis



Platform:  
Hardware + Runtime

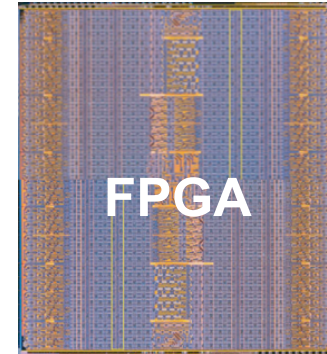
# Is High-Level Synthesis (using Lime) viable?



```

always @(posedge clk_22) begin
  if (reset_23) begin
    pc_30 <= 32'sd0;
    outReady_27 <= 1'd0;
    inReq_25 <= 1'd0;
  end else begin
    case (pc_30)
      0 : begin
        outReady_27 <= 1'd0;
        if (inReady_24) begin
          inReq_25 <= 1'd1;
          pc_30 <= 3'd1;
        end
      end
      1 : begin
        inReq_25 <= 1'd0;
        pc_30 <= 3'd2;
      end
      2 : begin
        a_31 <= inData_26[31:0];
        inReq_25 <= 1'd0;
        pc_30 <= 3'd3;
      end
      3 : begin
        multTmp_32 <= 32'sd2 * a_31;
        pc_30 <= 3'd4;
      end
    end
  end
end

```



*Using a high-level language, can a developer design hardware whose quality matches hardware designed with standard tools?*

YES!



NO!

# An Experiment



The screenshot shows a web browser window displaying the ICFPT 2013 Design Competition website. The browser's address bar shows the URL [lut.eee.u-ryukyu.ac.jp/dc13/](http://lut.eee.u-ryukyu.ac.jp/dc13/). The website header features a 14x14 grid with a pattern of yellow and pink squares on the left, and the text "ICFPT2013 Design Competition" and "Blokus Duo: Dec.2013, Kyoto Japan" in large white font on a dark blue background. Below the header is a navigation menu with links for "Main", "Rules", "Tools & Boards", and "Resources", along with a link to the "ICFPT 2013 Website". The main content area has a heading "ICFPT 2013 Design Competition: Blokus Duo" and a "News" section. The news section contains a list of updates, including a "DEADLINE EXTENSION" for contest paper submissions, and test & host updates. A "Show/Hide older news" button is located to the right of the news list. Below the news section is an "Introduction" section, which explains that the competition is for FPGA implementations of the Blokus Duo game, played on a 14x14 grid board with 21 different-shaped tiles.

ICFPT 2013 Design Competition  
Blokus Duo: Dec.2013, Kyoto Japan

[Main](#) | [Rules](#) | [Tools & Boards](#) | [Resources](#) | [ICFPT 2013 Website](#)

## ICFPT 2013 Design Competition: Blokus Duo

### News

- Sep.19: **DEADLINE EXTENSION: Contest paper / abstract submission will be extended to SEP.27,2013.**
- Sep.11: Test & host update in [Tools & Boards](#). Terminate code fix, etc.
- Sep.11: We're planning a poster session on competition designs.
- Aug.22: Entry/submission is now open! The "Getting Involved" section of this page has a major update.
- Aug.14: Test & host update in [Tools & Boards](#). Terminate code fix, etc.
- May.14: Test & host update in [Tools & Boards](#). Linux/Cygwin compatibility fix.

[Show/Hide older news](#)

### Introduction

Following on the successful Connect6 competitions, we'll compete against each other at a game called Blokus Duo with our FPGA implementations, at [ICFPT 2013](#).

Blokus Duo is a two-player game played on a square, 14x14 grid board. Each player has 21 different-shaped game tiles. Each new piece played must be placed to contact at least one piece of the same color with their corner-to-corner. Edge-to-edge contact is only allowed to the other color.

# Blokus Demo



Lime

# The Liquid Metal Programming Language



## What is Lime?

**Lime = Java + Isolation + Abstract Parallelism**

**Isolation: ability to move computation**

**local** keyword

Immutable types

**Abstract Parallelism: freedom to schedule**

Stream programming model (**task**, =>)

Data parallel constructs (@ map, ! reduce)



# The Lime Development Experience

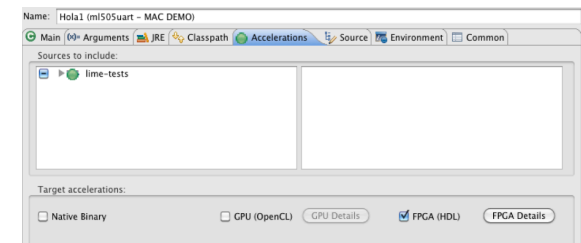
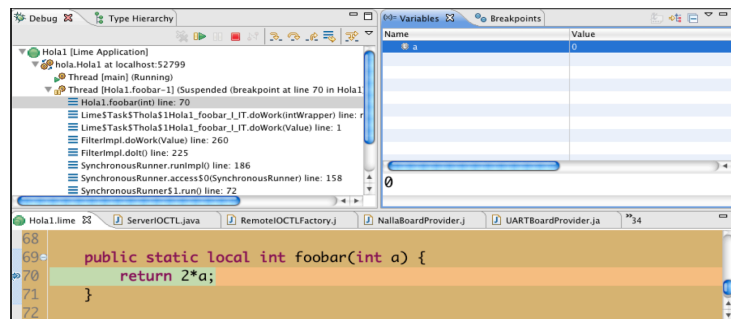
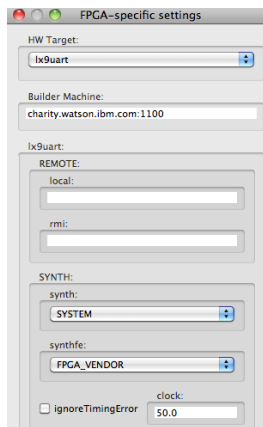
## 1. Prototype in standard Java

- Lime is a superset of Java
- **Java-like Eclipse IDE (editors, debugger, navigation)**

## 2. Gentle, incremental migration to parallel Lime code

Lime language constructs restrict program to safe parallel structures

- Isolation
- Immutability
- Safe Deterministic Parallelism
- Bounded Space



# Implementing Blokus player in Lime

```
public interface Player {
    /**
     * Return some descriptive name for this Player
     */
    public string getName();

    /**
     * Reset the state of the game to g.
     */
    public void reset(Game g);

    /**
     * What is the next move, after the opponent makes move m?
     */
    public Move nextMove(Move m);
}
```

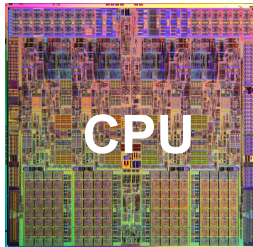
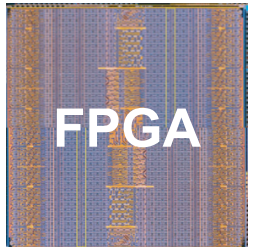
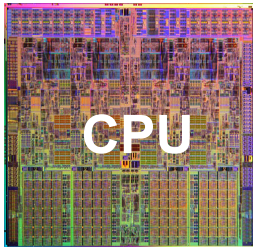
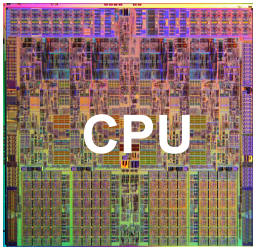
Player is stateful object

Relocation Brackets `[ ( ) ]`  
indicate tasks that  
can relocate to device

**Task graph** expression specifies coarse-grain dataflow:

Connect operator `=>`  
abstracts communication

```
source() => ([ task new BlokusPlayer().nextMove ]) => sink()
```





Lime for a standalone (headless/no host) deployment

```
([ task receiveMove => task move => task sendMove ])
```

Entire task graph inside relocation brackets

## Lime Native Interface

```

import lime.lang.annotations.lni.InputPins;
import lime.lang.annotations.lni.OutputPins;

public class NativeBlokus {

    @InputPins({"opp_move:72", "opp_rdy"})
    @OutputPins({"opp_ack"})
    public static native local byte[[9]] receiveMove();

    @InputPins({"lime_ack"})
    @OutputPins({"lime_move:32", "lime_rdy"})
    public static native local void sendMove(byte[[4]] b);

}

```

Implement "native" methods in  
VHDL/Verilog/etc.

## Blokus AI Algorithm

### Minimax tree search

- Based on board **evaluation function**

### Fixed budget (1 sec response)

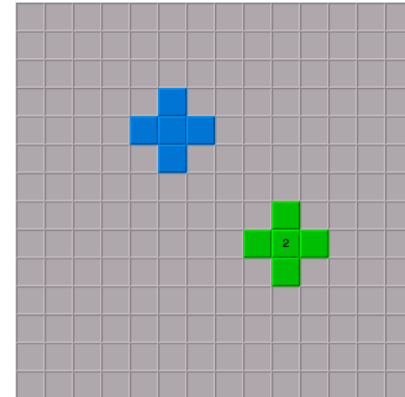
- Iteratively deepen tree until budget exhausted

### Maximum bounded tree size

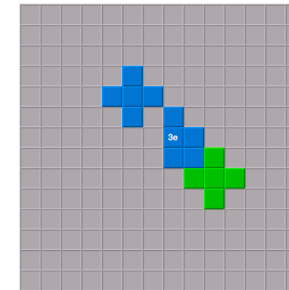
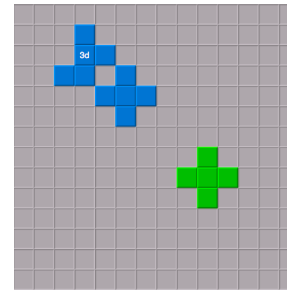
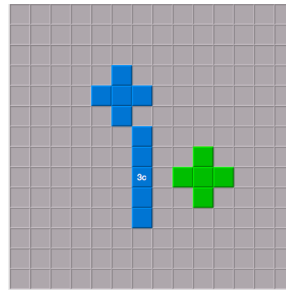
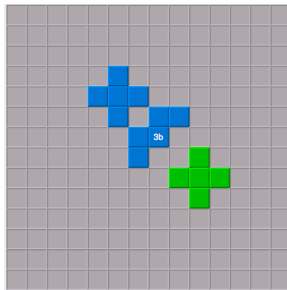
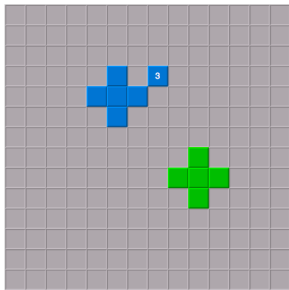
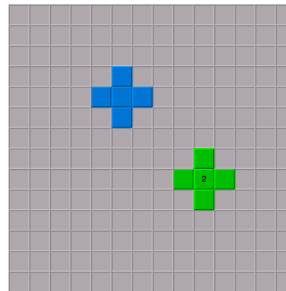
- Statically allocated data structures – no dynamic memory allocation

### Relatively simple algorithm for software AI players

- Relatively complicated data structures for hardware implementation

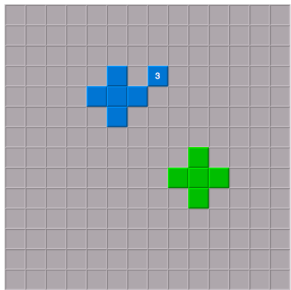
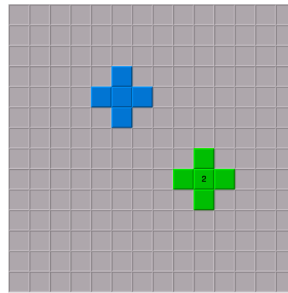


# Minimax Step 1: Enumerate all possible moves

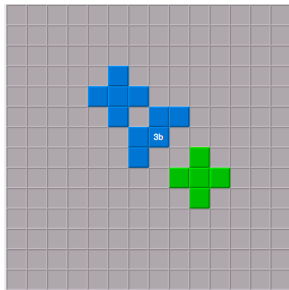


...

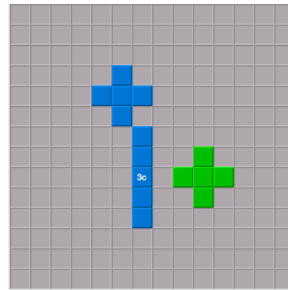
## Minimax Step 2: Score each move with *evaluation function*



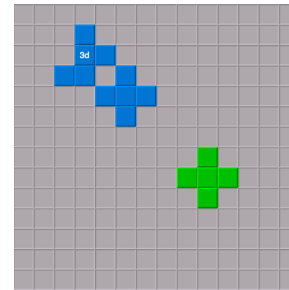
100



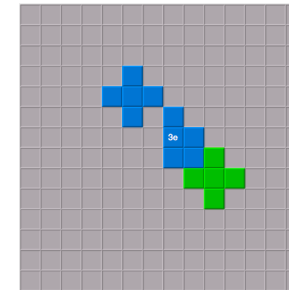
700



600



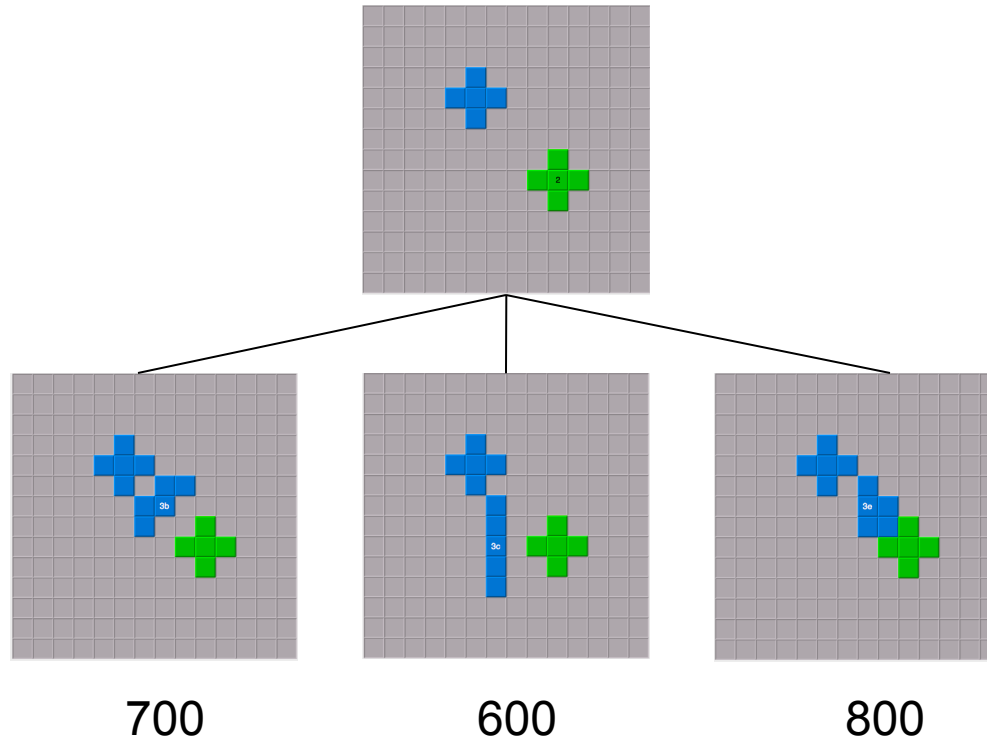
300



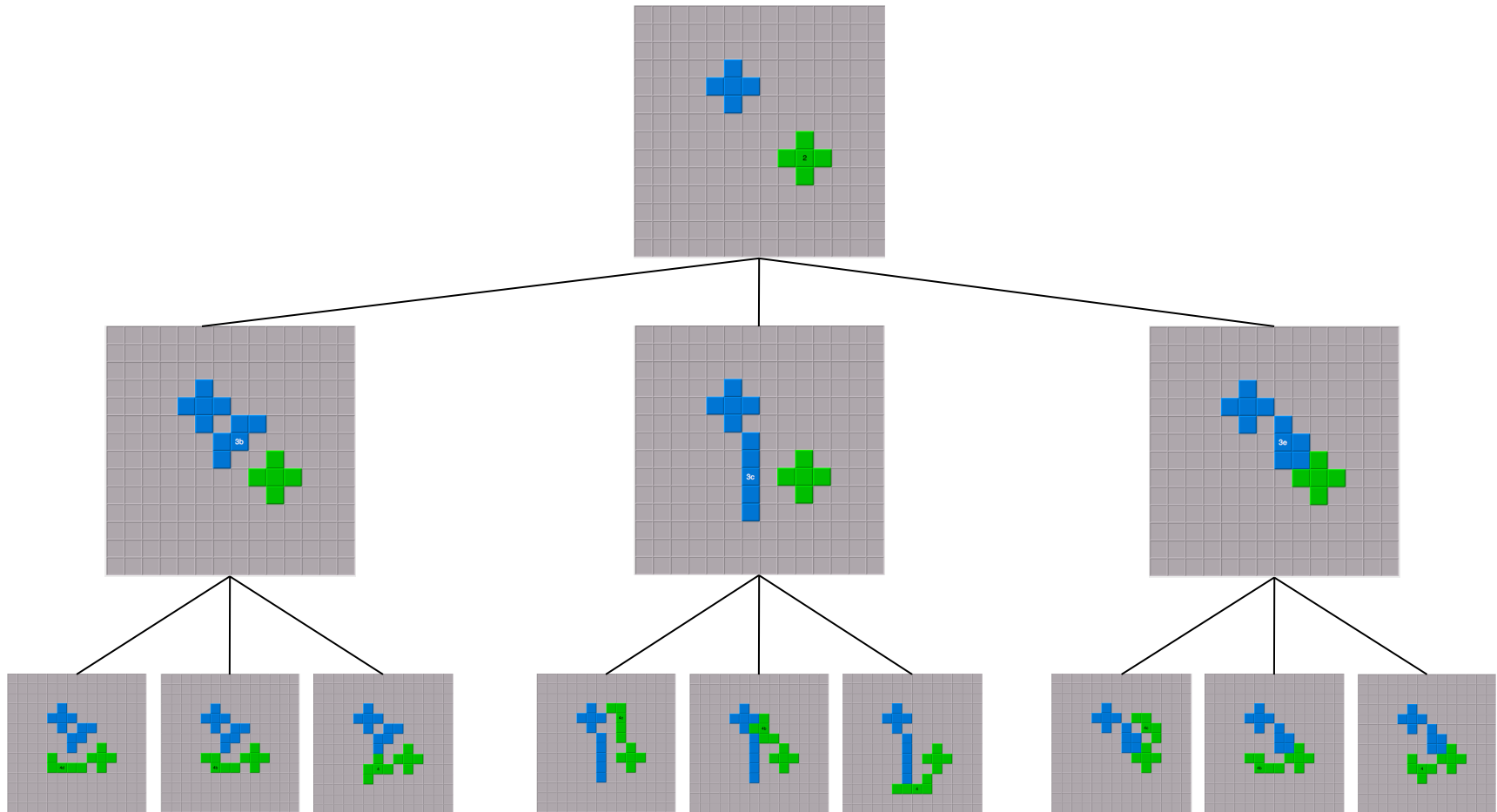
800

....

## Minimax Step 3: Pick best $k$ moves and add to search tree

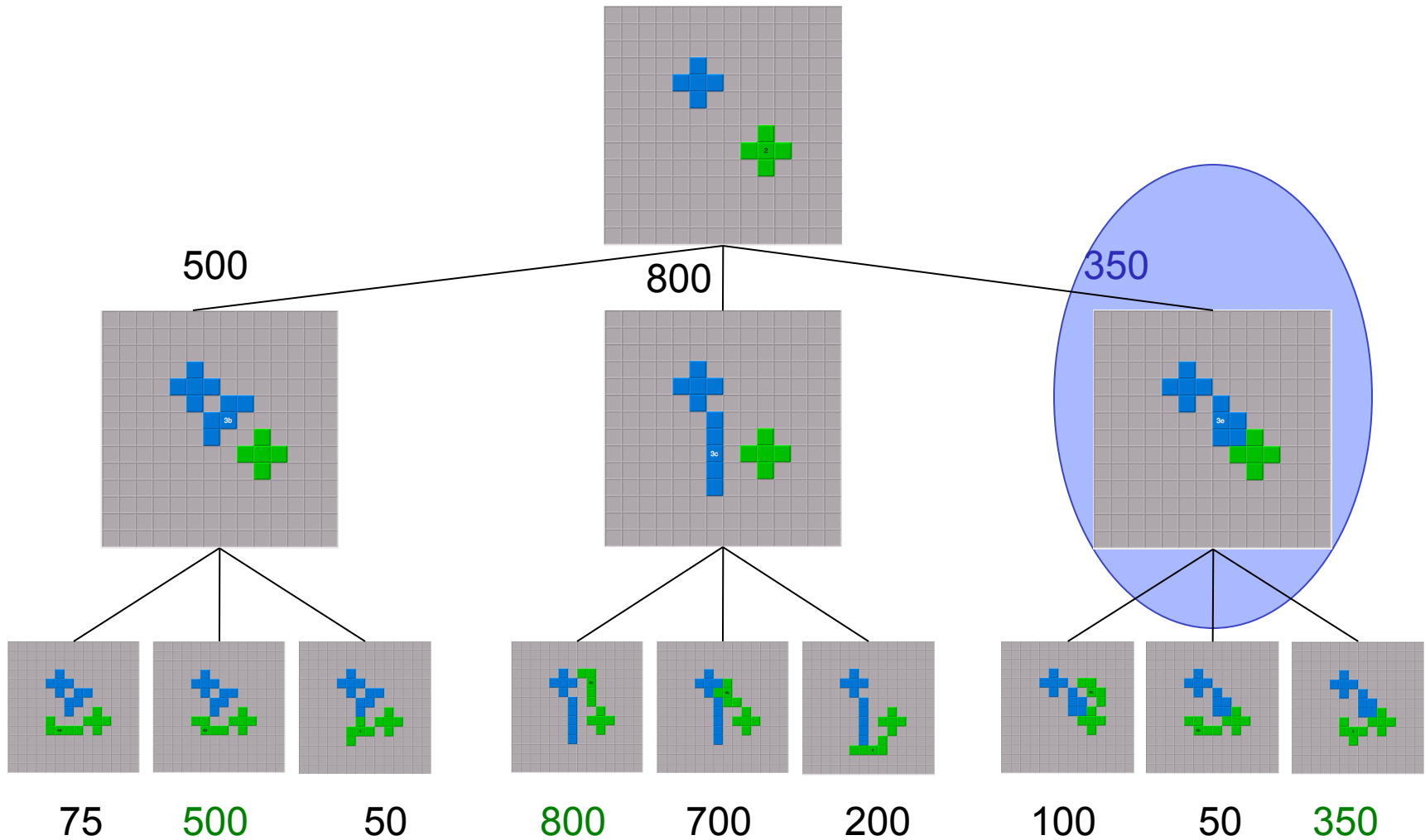


### Minimax Step 3: Iterate until budget exhausted

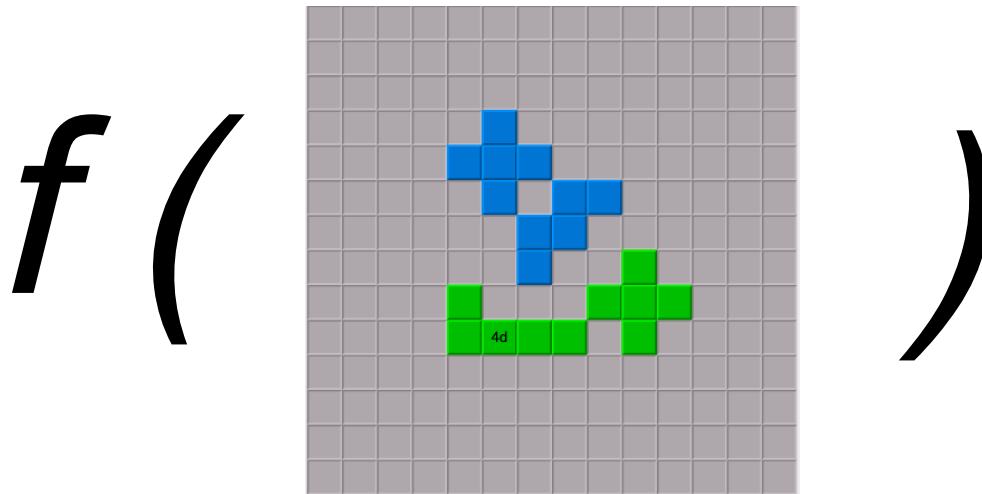


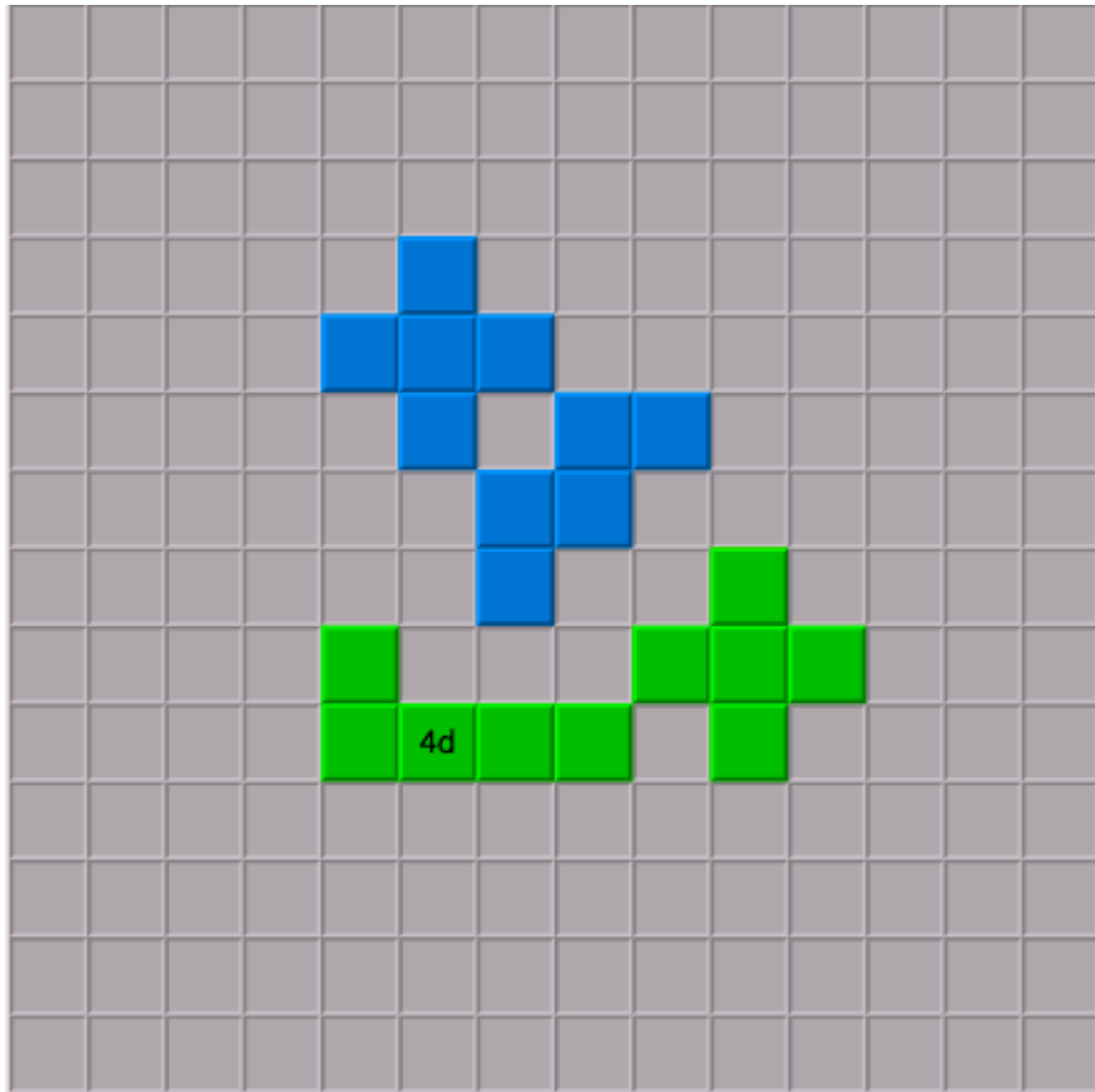


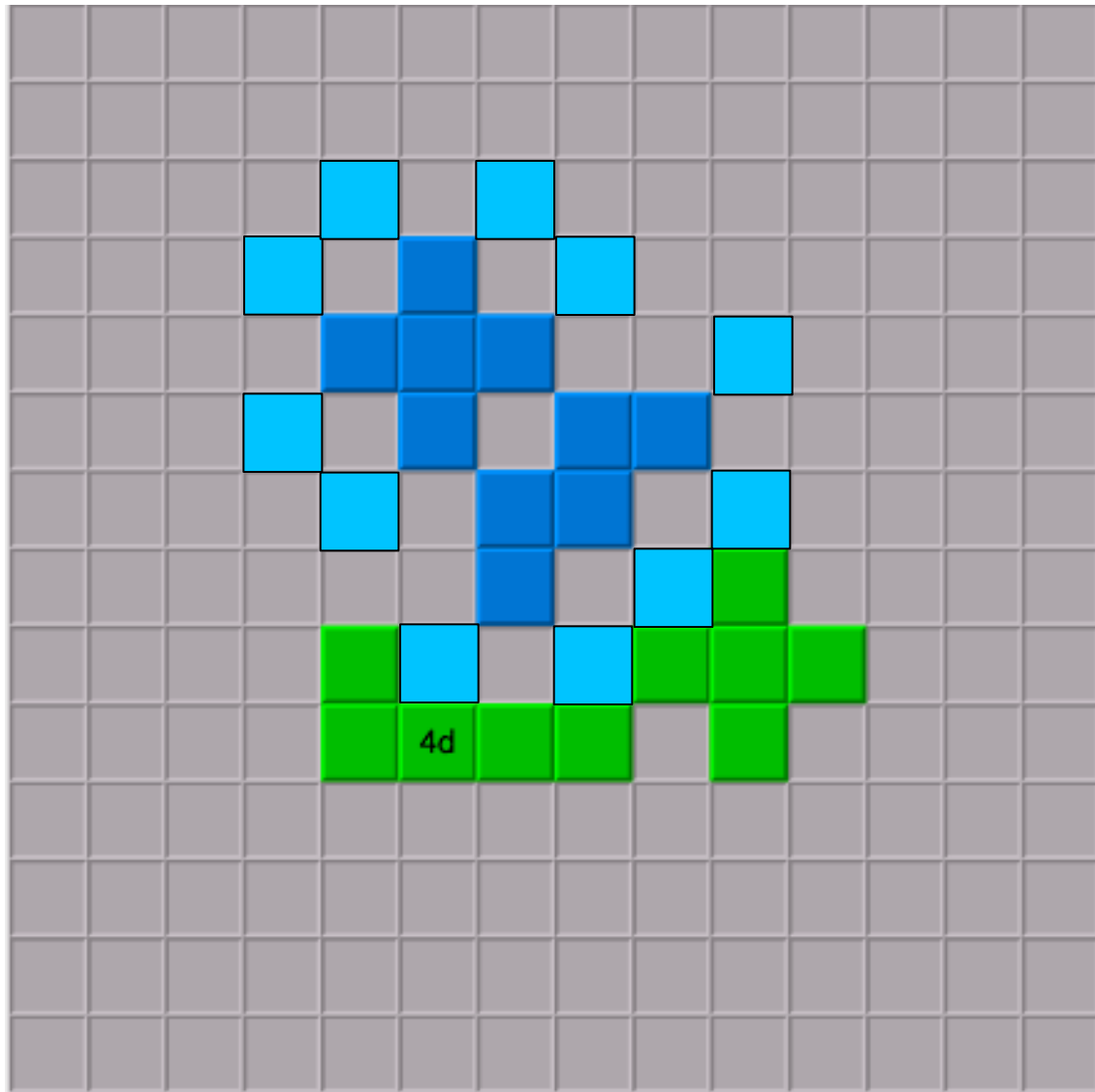
# Minimax Final Step: Choose best move via minimax reduction

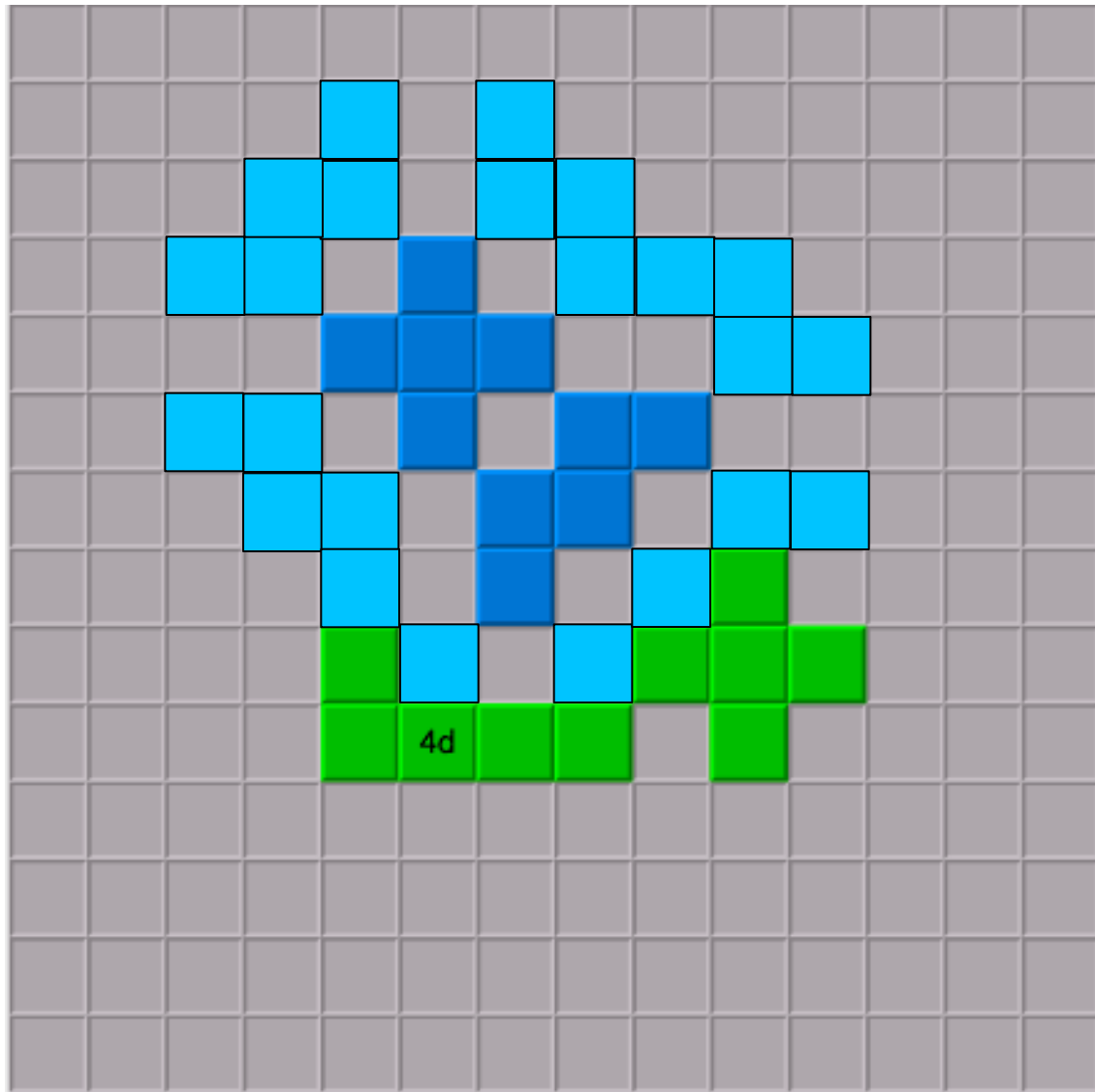


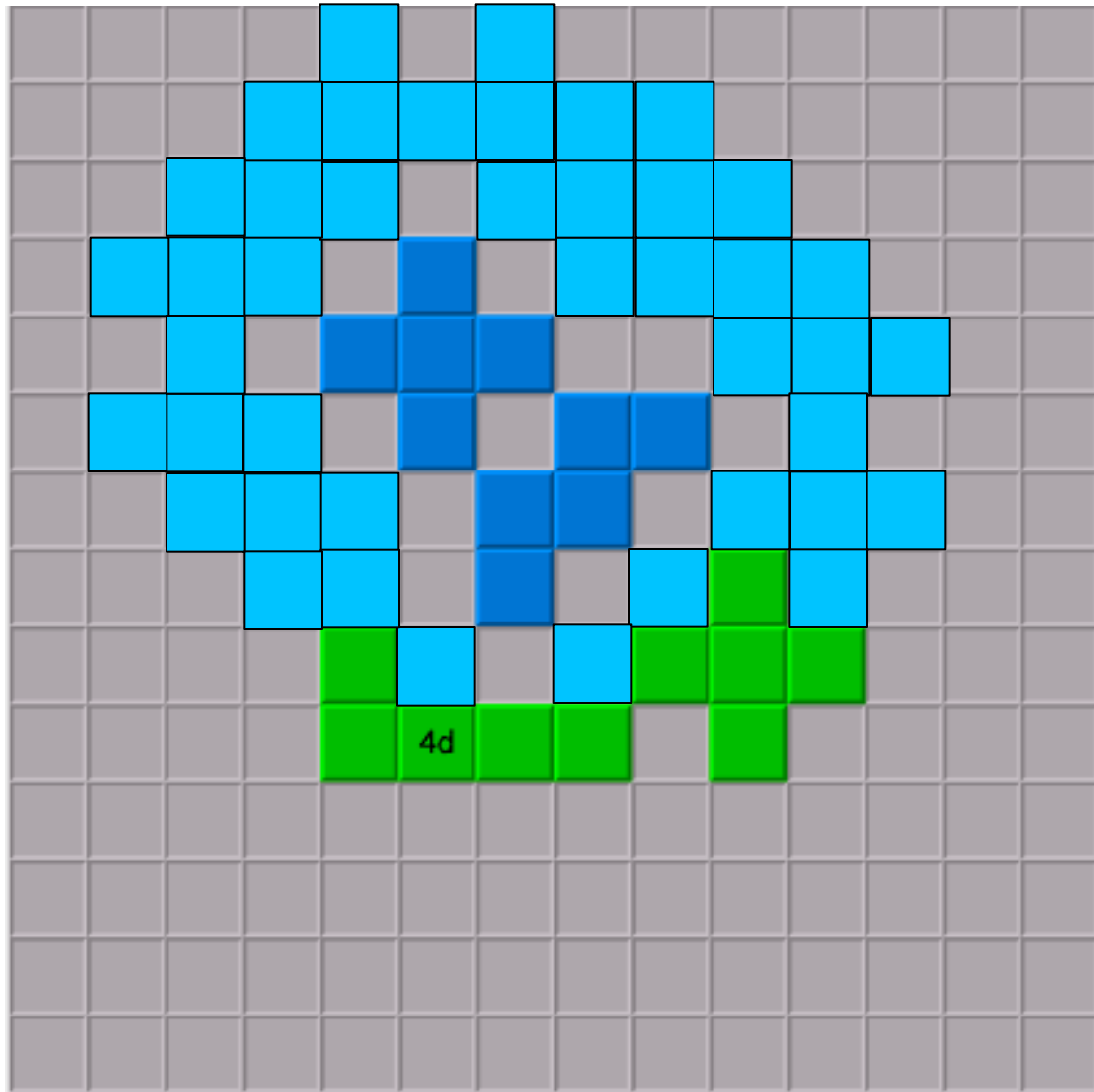
# Territory-Based Evaluation Function

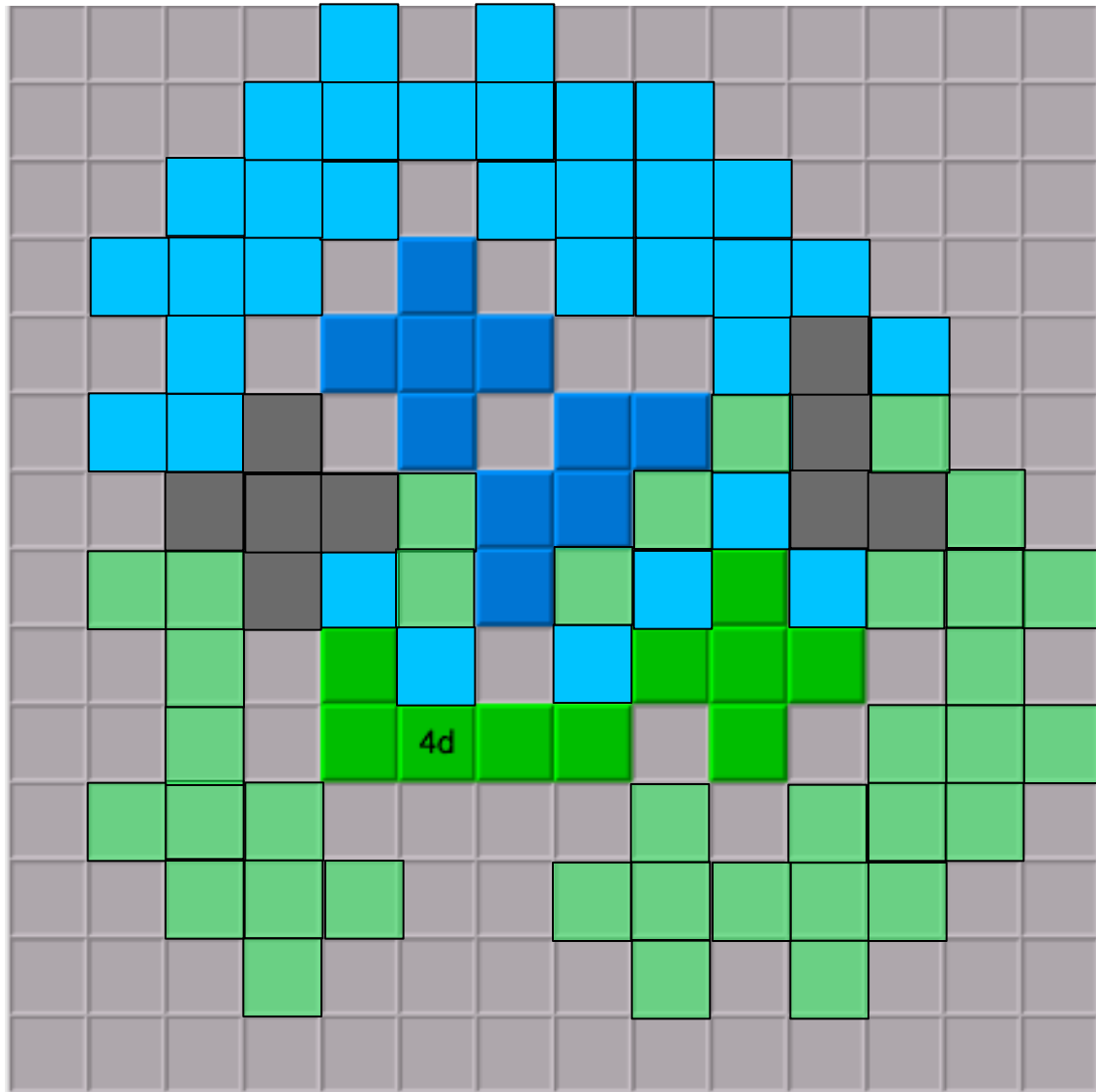












## Inner loop of evaluation function in Lime

local qualifier guarantees isolation

value classes (programmable primitives)

bounded, immutable arrays

bounded integers

```

public static local Row[[14]] manhattanNeighbors(Row[[14]] g) {

    Row[14] r = new Row[14];
    Row upper = 0;
    Row mid = g[0];
    for (int<14> i){
        Row lower = (i == 13) ? (Row)0n : (Row)g[i+1];

        // manhattan propagation.
        r[i] = (Row)(upper | lower | (mid << 1) | (mid >>> 1));

        // update rows.
        upper = mid;
        mid = lower;
    }

    return new Row[[14]](r);
}

```

Lime type system invariants allow efficient translation  
inner loop – potentially one cycle



# Implementation



## ML505 / Xilinx-Virtex 5 LX50T

7200 slices  
48 DSP48E  
2160 kB BRAM

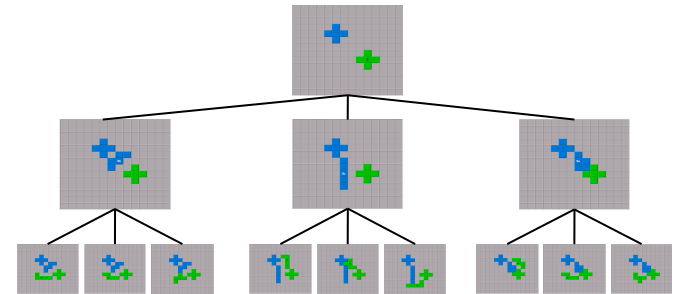
### Design Stats

Line Lines of Code	4231
Lines of Verilog generated	24,657
Lines of hand-written Verilog	186
Frequency	85 MHz
LUTs	15,917 (55.3%)
Flip Flops	11,050 (38.4%)
18kB BRAMs	64 (54.2%)
DSPs	14 (29.2%)

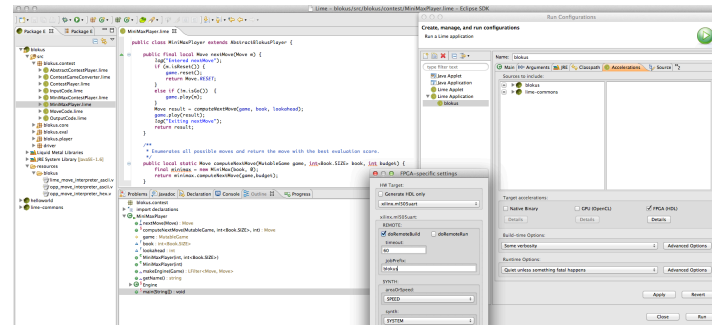
# Lessons Learned So Far

AI Search: Ideal domain for high-level synthesis

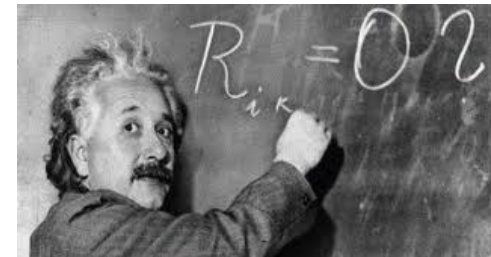
- Algorithmic tuning
- Iterative development cycle
- High computation/communication ratio



Lime development cycle rocks



Software engineer can implement complex hardware in Lime



*Using a high-level language, can a developer design hardware whose quality matches hardware designed with standard tools?*



*Can the Lime team design a Blokus player in Lime that can win the 2013 ICFPT Design Competition?  
Tune in early December!*

