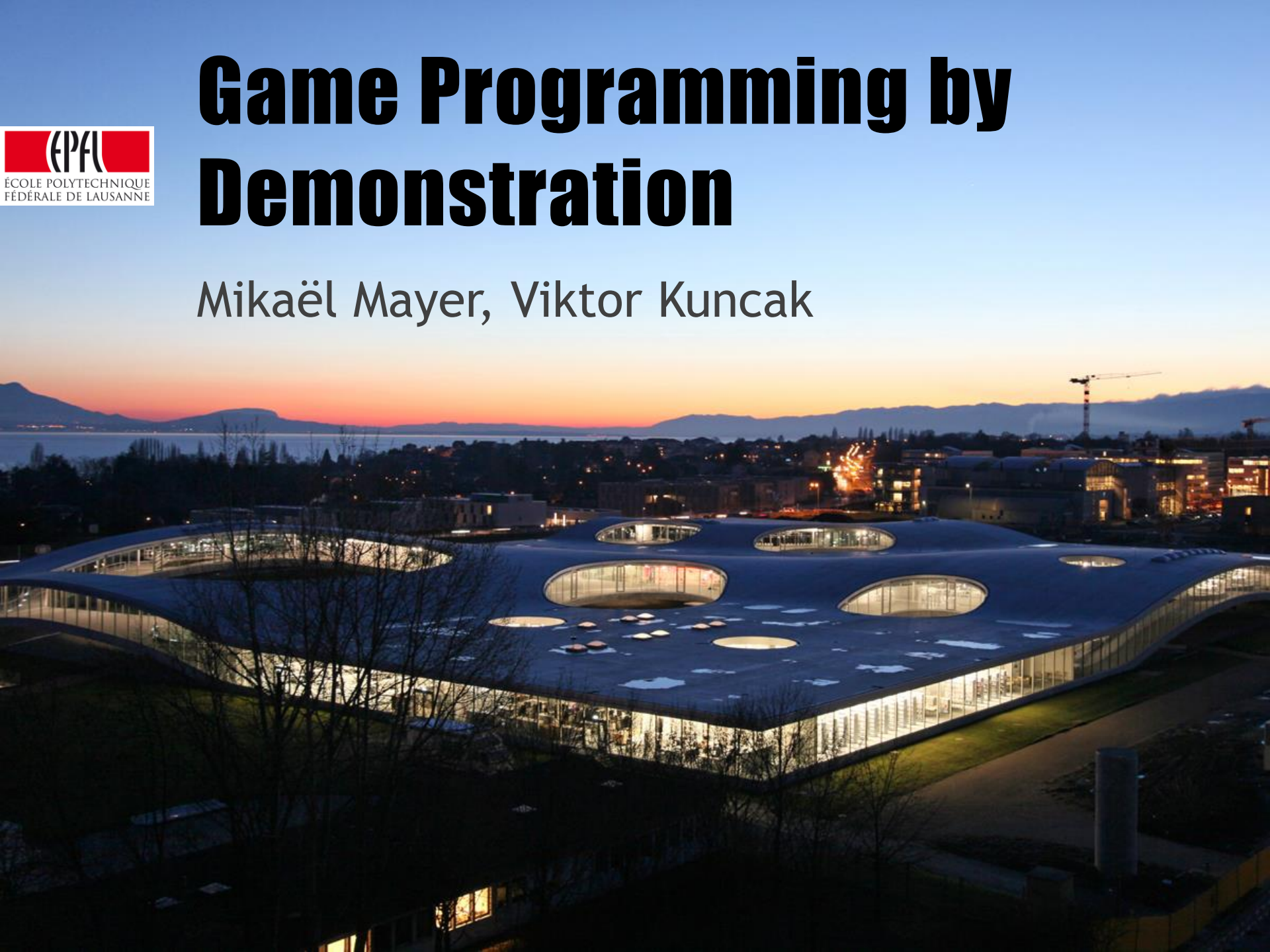
Game Programming by Demonstration

Mikaël Mayer, Viktor Kuncak



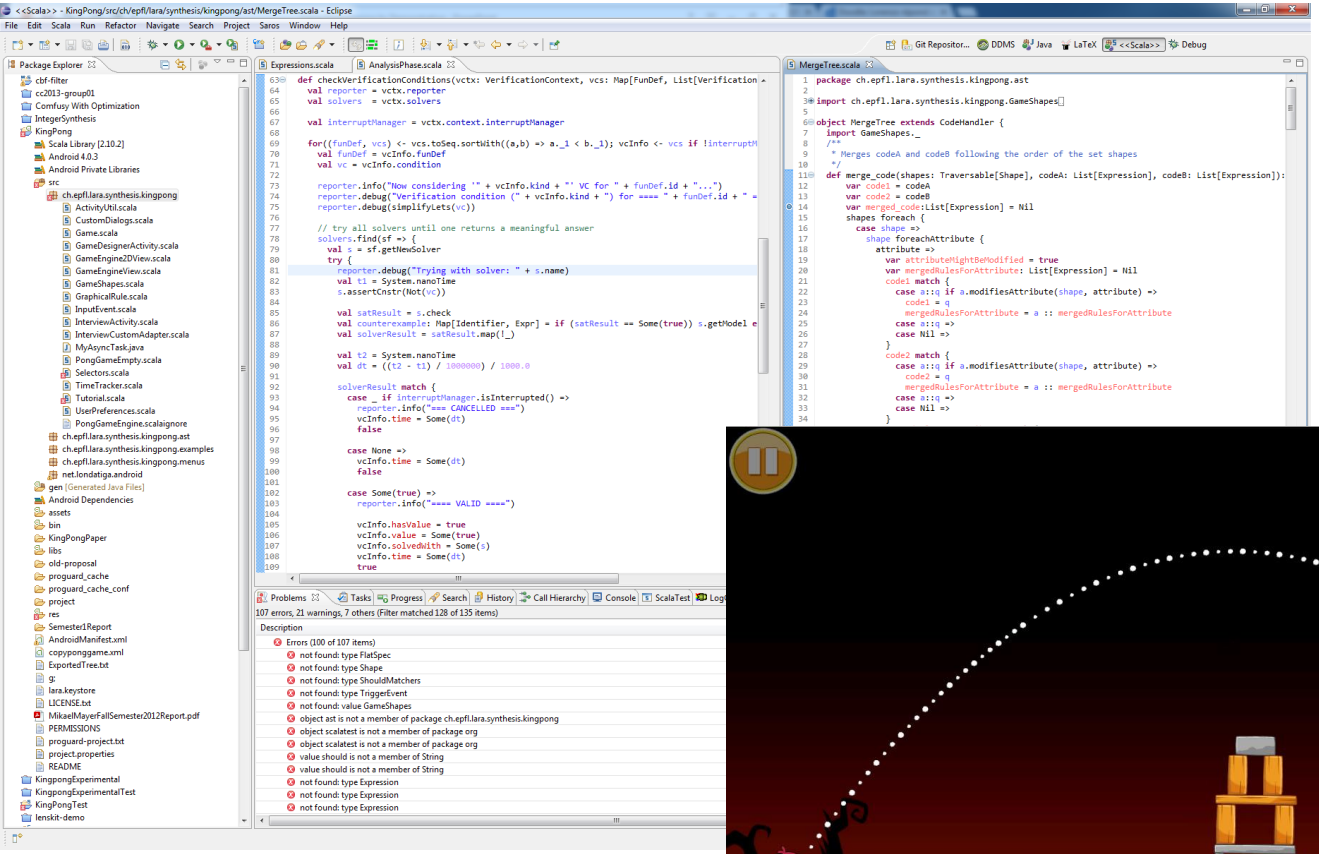
How to introduce programming to new users?

Motivation

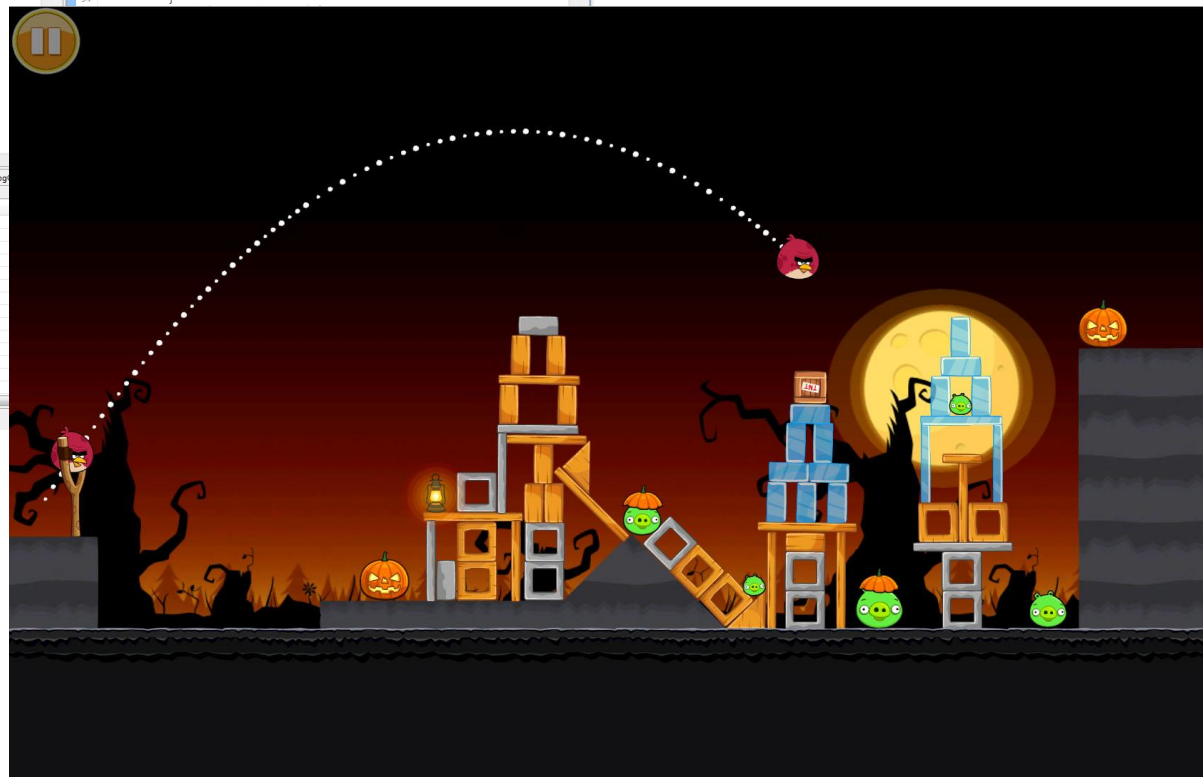
- Millions of smartphone users and gamers
- Few are programmers - because it is hard
  - requires the programmer to learn complex APIs
  - involves debugging, which is time-consuming
  - disconnect between the code and the game

- How to make programming more accessible ?

What is more accessible ?



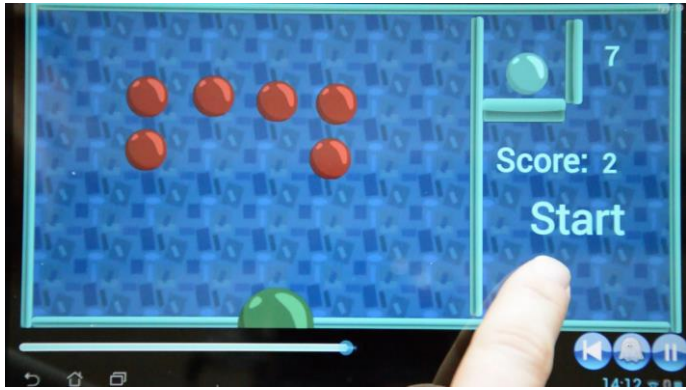
versus

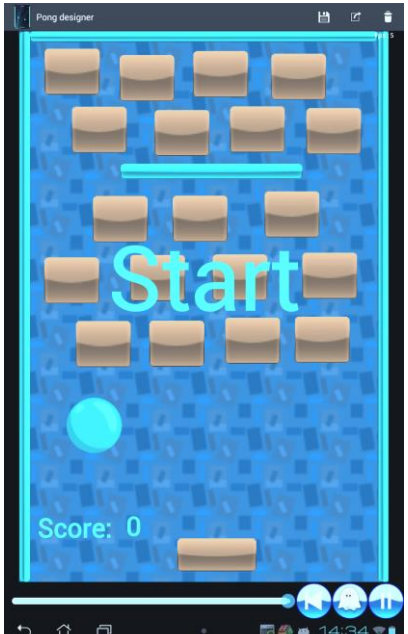


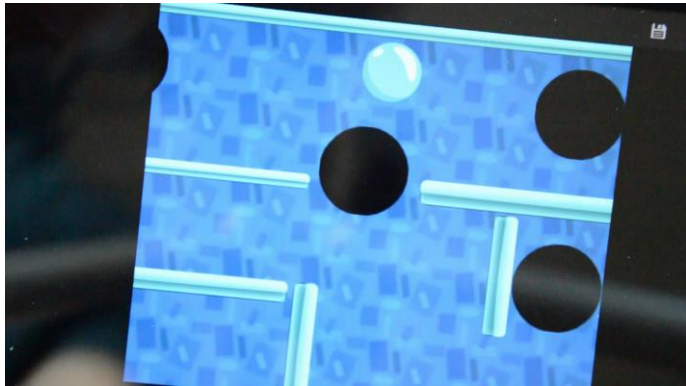
Pong Designer : our approach for game programming by demonstration.

Games developed

- Pong
- Brick Breaker
- Pacman
- Space invaders
- Pool
- Maze
- ….

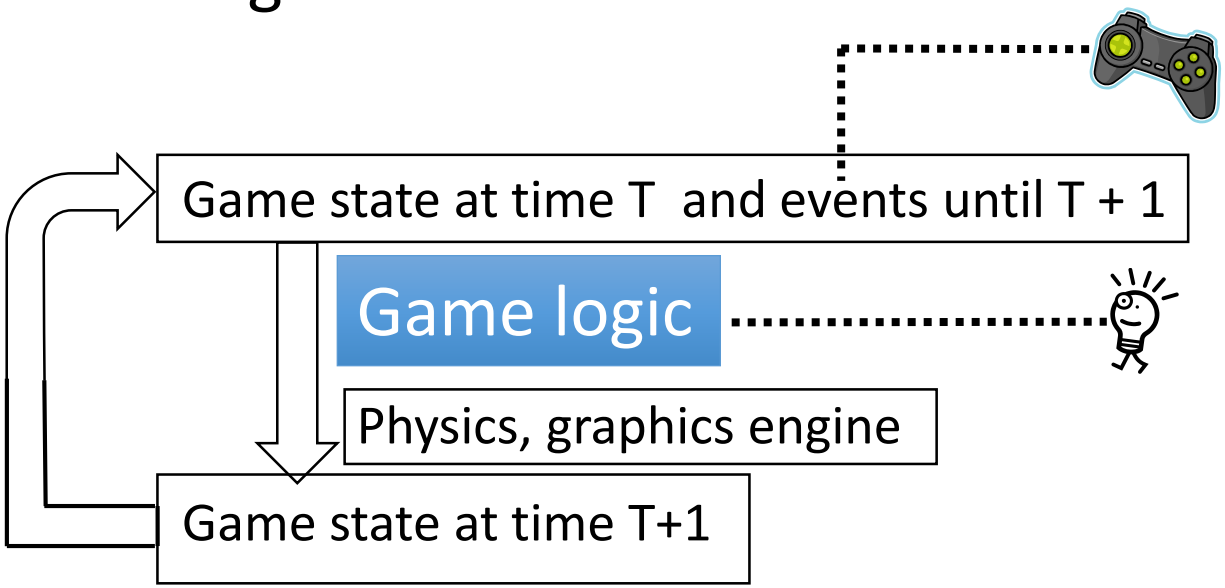






Game programming

- Game engines:



Game logic is conventionally written in languages like C++, Flash, Java, Scala

Conventional game programming

Game logic

1. Find out conditions for the rule to apply .....
   - Object-specific, triggers, events, etc.
2. Write code to run when these conditions hold.
   - Modify state. .....
3. Rewrite other rules to comply with the new rule.
   - Run, debug after playing .....

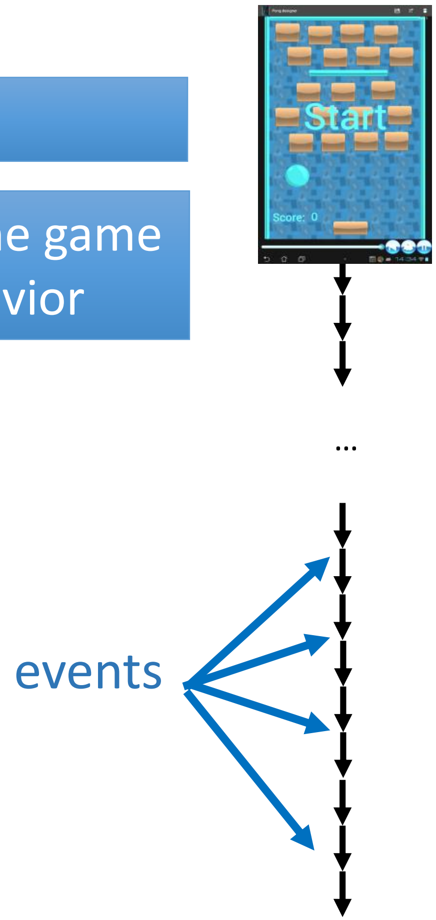
Pong Designer Approach

Can we do the same by demonstration?

Pong Designer Approach

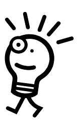
Create initial state

Continuously run the game and refine the behavior



- Create initial state
- Start the game and see how it evolves
  - Default behaviors apply

Demonstrating game logic

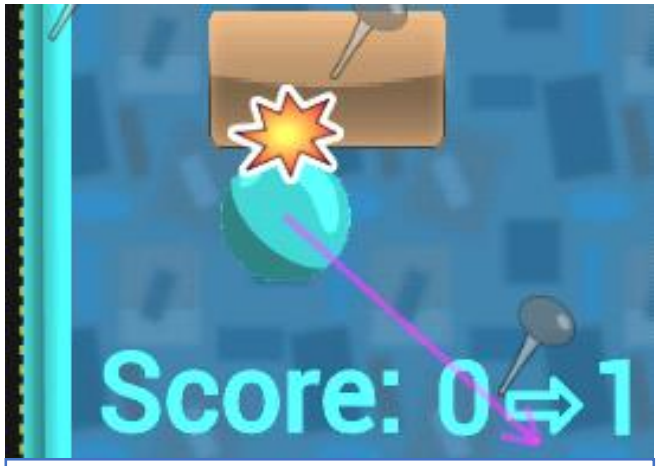
1. Pause the game
2. Prepare game state
   - Previous time 
   - Or arrange objects
3. Select events 
4. Change state 
5. Validate
   - System infers rules
   - Manual modification
6. Repeat 

Main techniques

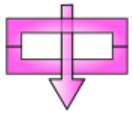
- Touch-based interface
- Access to 5 seconds history
- Visualization and modification of everything
- **Automatic rule inference**
- Incremental addition of demonstrations

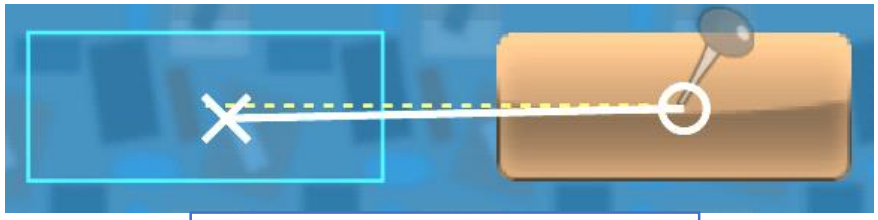
Changes are visual

- Game state

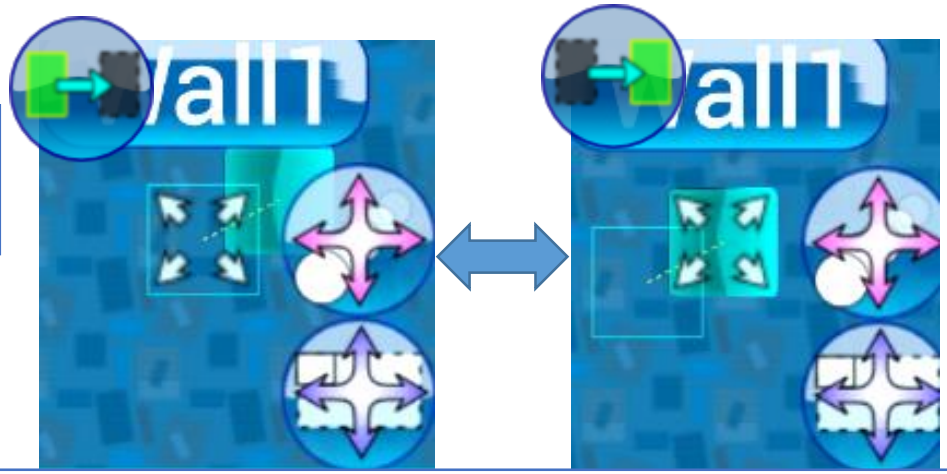


Change numbers, text, color, speed, position





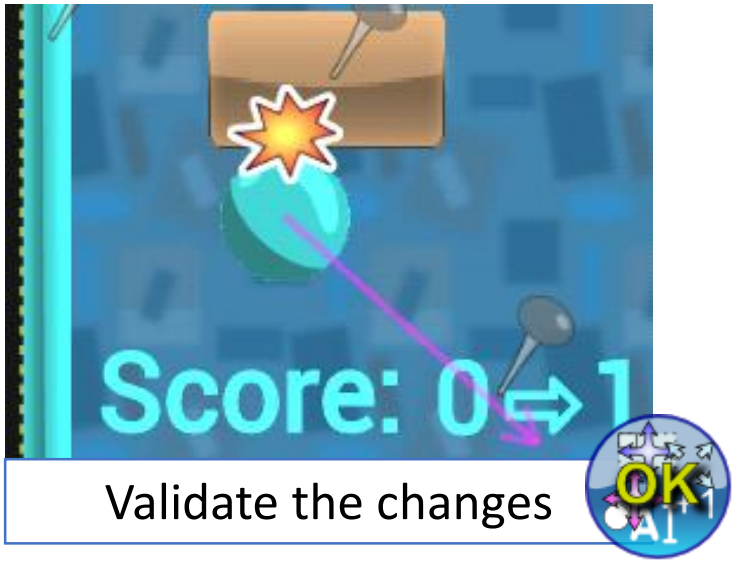
Events are visible



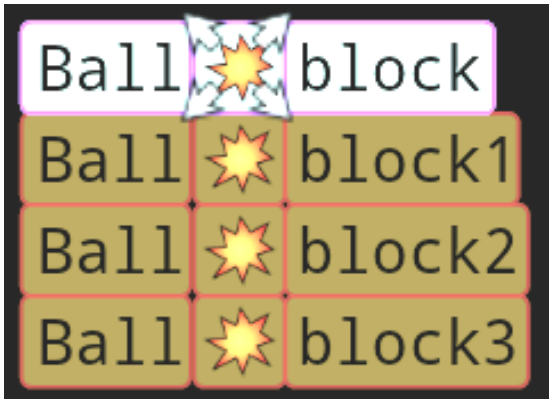
Input and output are both modifiable

Inferring rules

- Accept the changes



Validate the changes



The rule is automatically inferred

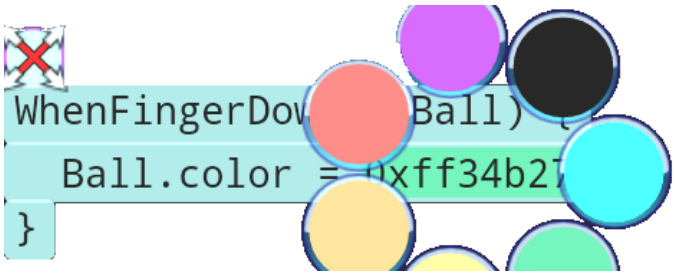
```
Ball block
WhenCollisionBetween(Ball, block) {
  block.visible = false
  Score.value = Score.prev_value + 1 // <-|->
}
```

The user can modify the rule and constants

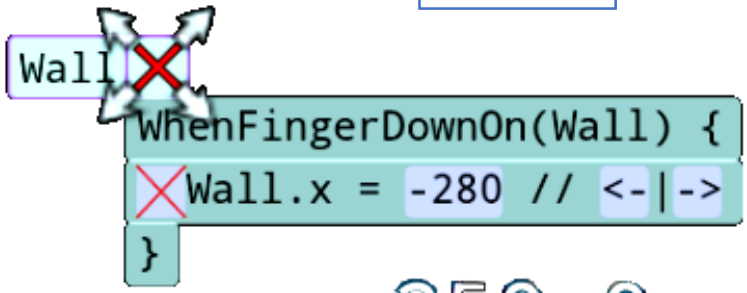
Code is interactive

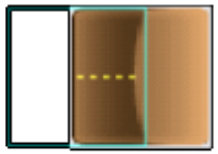
- Changing constants shows the effects in the game

Colors

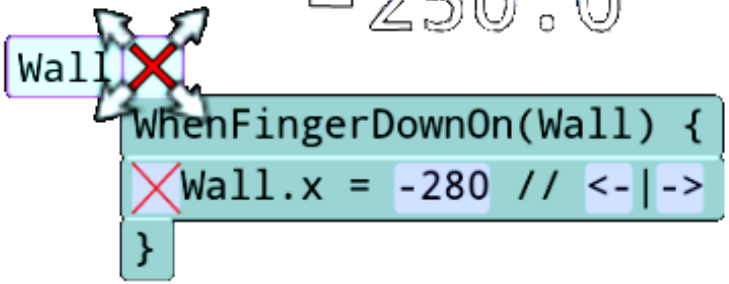


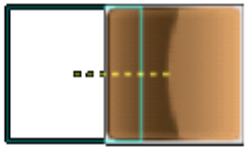
Position



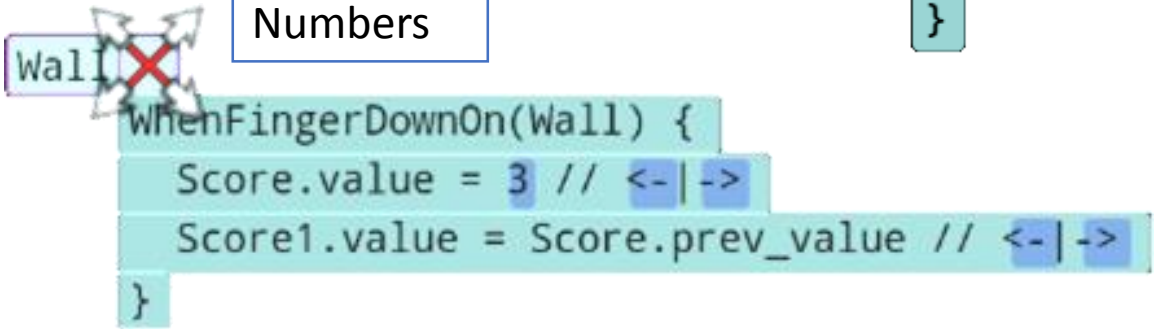


-250.0

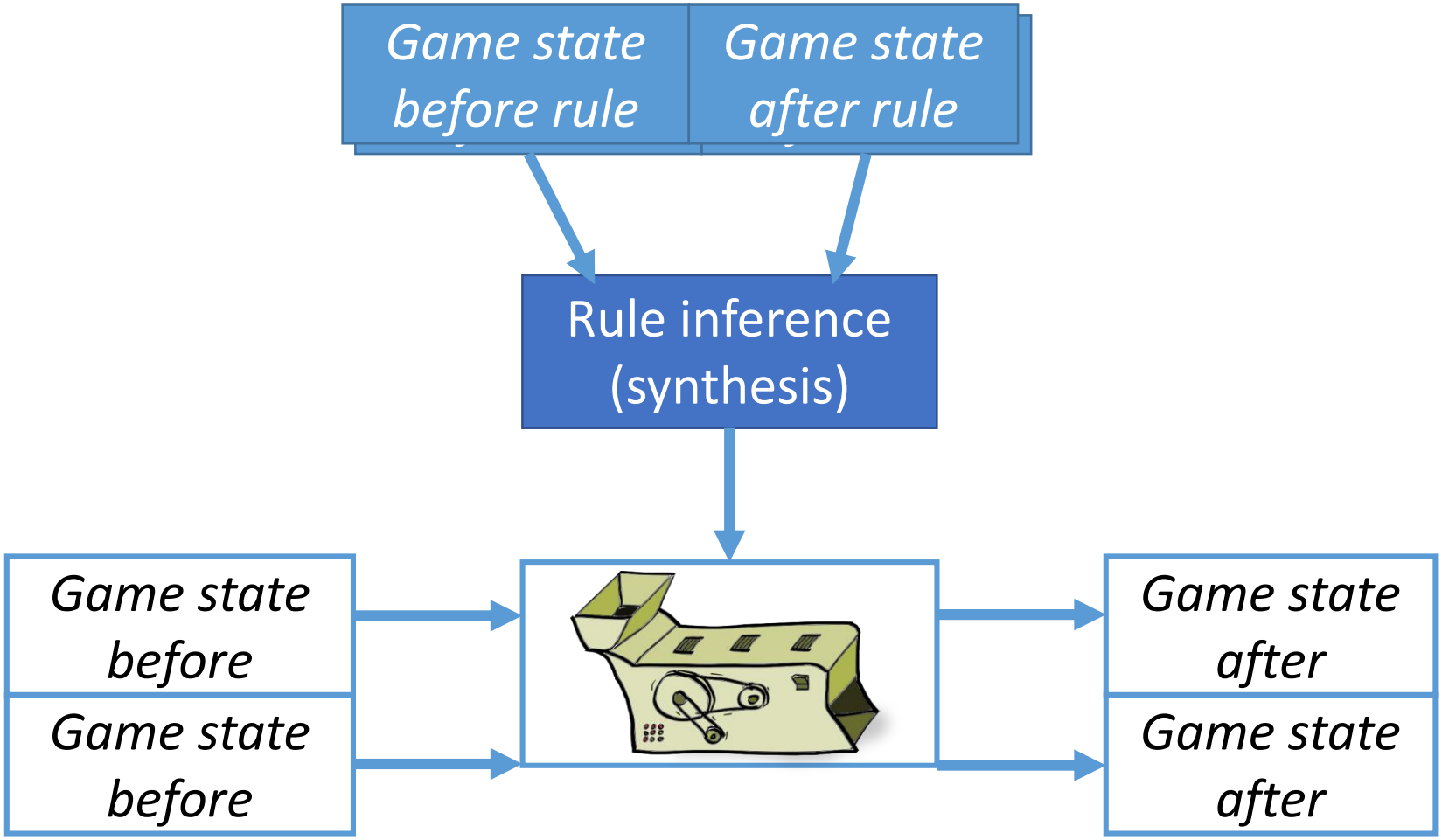




Numbers



How infer rules?



Templates inferring rules

- Generalizing from input/output examples:

If block.x ≠ block.prev_x

« block.x = block.prev_x + {block.x - block.prev_x}

|| block.x = {block.x}

|| block.x = 2*obj.x – obj2.x //for some obj, obj2

….»

Resolve the ambiguity by either providing a second example (implicit), or selecting the desired line of code (explicit).

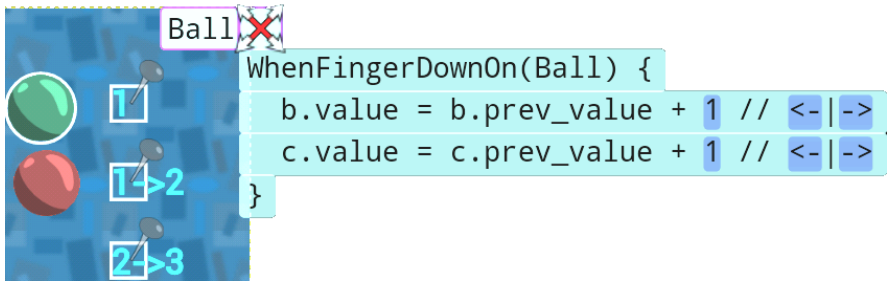
Template parameters

- Objects
  - Iterate through all, find which ones can explain the demonstration (alignment, result, etc.)
  - Iterate through pairs of objects (mirror, binary operations, etc.)
- Constants

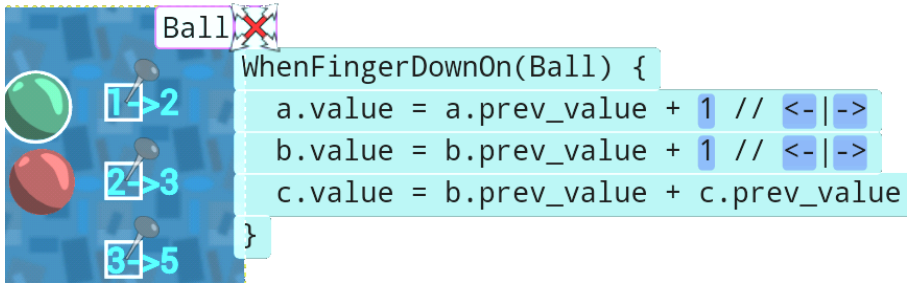
  (position, color, velocity, angle, text)
  - Approximate comparison.
  - Grid fit for angles and positions.

Accepting new examples

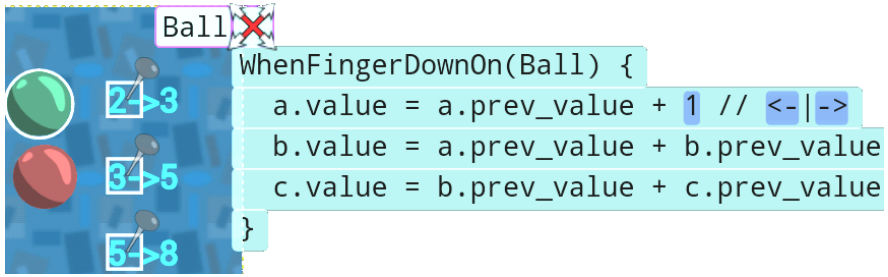
- Fibonacci through examples:



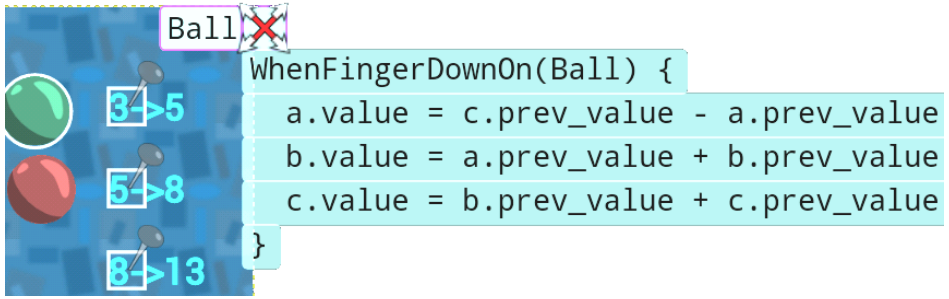
Fibonacci 1, 1, 2



Fibonacci 1, 2, 3



Fibonacci 2, 3, 5



Fibonacci 3, 5, 8

Syracuse sequence

- Clock as a ball
- Demonstrate n/2, n*3, n*3+1
- Test on n%2 to copy either n/2 or 3n+1
- Demonstrate appending number to “seq:”

Primes listing

- 2 balls
  - One to increment the test, the other to increment the quotient
- Demonstrate remainder with (11, 3) => 2
- Output if quotient greater than half
- Stop if remainder=0

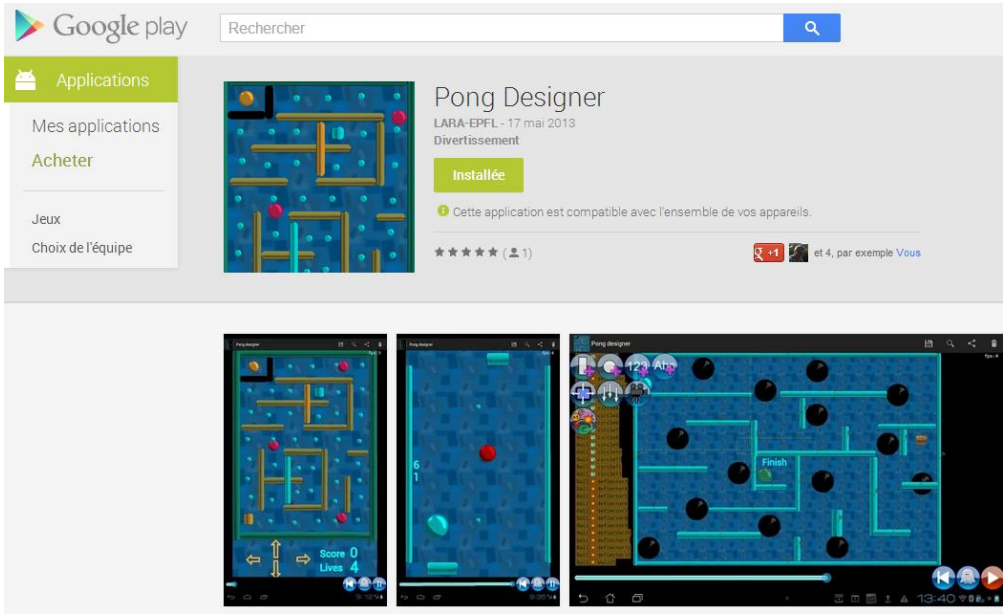
Minsky Machine

- Clock as a ball transferring the PC to read memory.
- Memory on a counter
  - A rule-per-integer-value increases or decreases registers and set up new conditional program counters
- Integers testing if registers are zero
  - Override program counter

Available on-line

lara.epfl.ch/w/pong

play.google.com/store/apps/details?id=ch.epfl.lara.synthesis.kingpong



New version is coming soon

Upcoming version of Pong Designer

- Better engine and interface
- Categories

```
For b in blocks:
  if ball collides b:
    b.visible = false
```

- Behaviors using constraints

X = Choose(x $\Rightarrow$ right $\leq$ border.left)

Existing approaches

- Accessible game programming
  - Scratch
  - Construct 2
  - Kodu
  - GameMaker
- Interactive programming environments
  - Khan Academy 2012
  - Kojo
  - Bret Victor, Inventing on Principle
  - TouchDevelop
- Learning from input-output examples
  - Automating String processing (Gulwani, 2012)
  - Marquise (Myers et al., 1993)
  - Behavioral Programming (Harel et al. 2012)

Conclusion

- Aim to bring game development to end users
- On-the-fly incremental rule demonstration
- **Automatic rule inference**
- Touch-based interface
- Access to history
- Visualization and modification of everything
- Freely available working implementation on Android

lara.epfl.ch/w/pong


fps:6
Questions?
baaabc
a
b
c