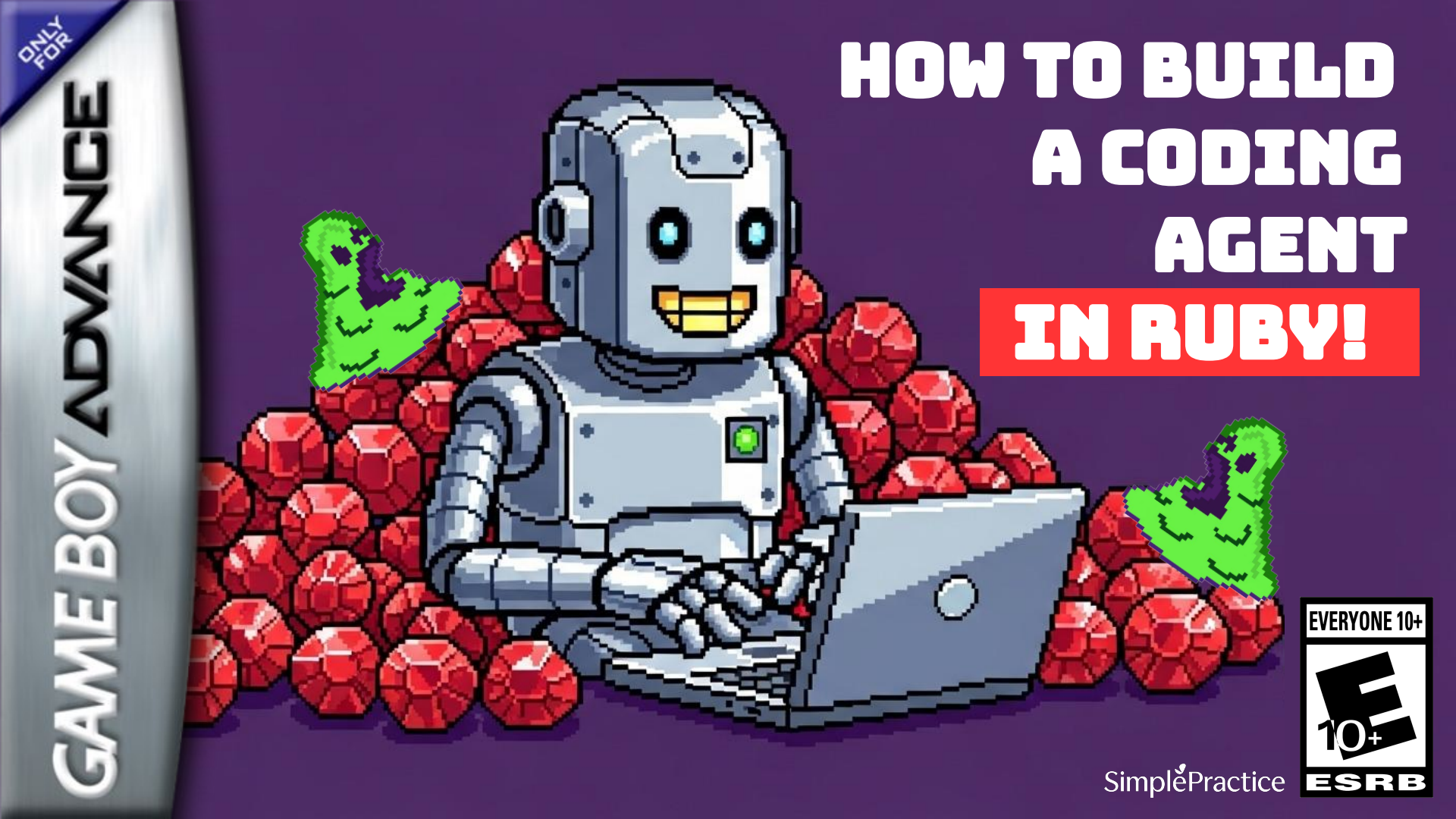

ONLY FOR
GAME BOY ADVANCE
HOW TO BUILD A CODING AGENT
IN RUBY!
EVERYONE 10+
10+
ESRB
SimplePractice

Demo

tinyurl.com/sp-agent

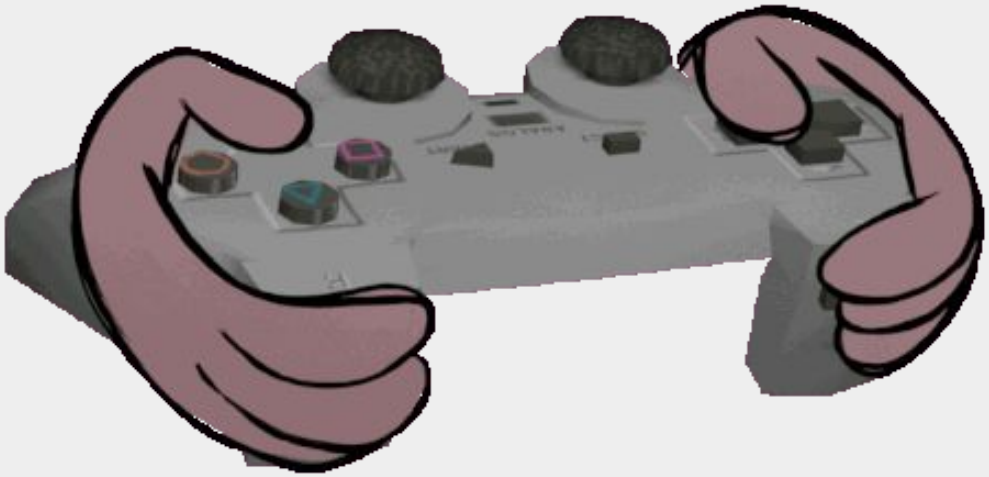
Why care about Agents?

SimplePractice

State Machine

SimplePractice

SimplePractice



SimplePractice

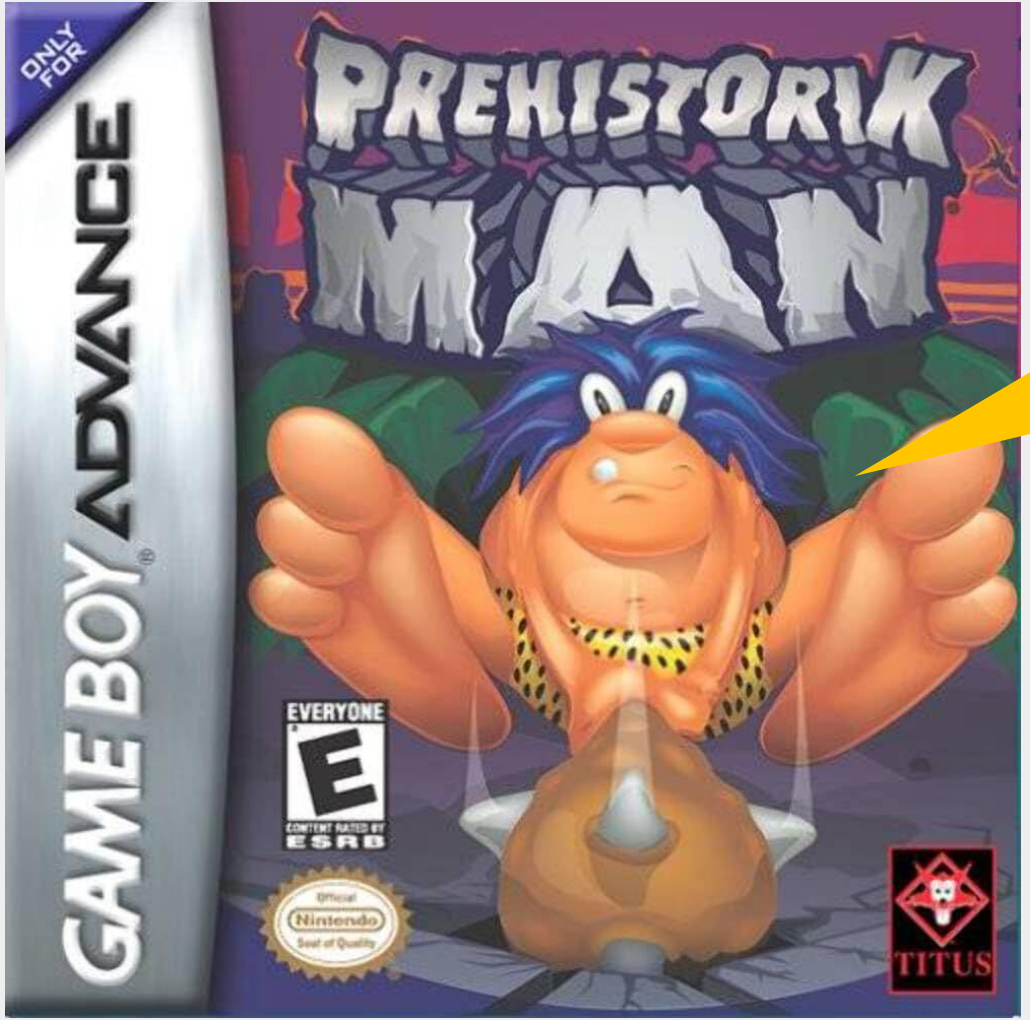
User watches machine

Machine controls machine

SimplePractice

“When you combine ignorance and leverage, you get some pretty interesting results.”

- Warren Buffett



SimplePractice

Toolformer: Language Models Can Teach Themselves to Use Tools

Timo Schick **Jane Dwivedi-Yu** **Roberto Dessì**[†] **Roberta Raileanu**
Maria Lomeli **Luke Zettlemoyer** **Nicola Cancedda** **Thomas Scialom**

Meta AI Research [†]Universitat Pompeu Fabra

SimplePractice

LLMs learn when to use tools

SimplePractice

REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao[*,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

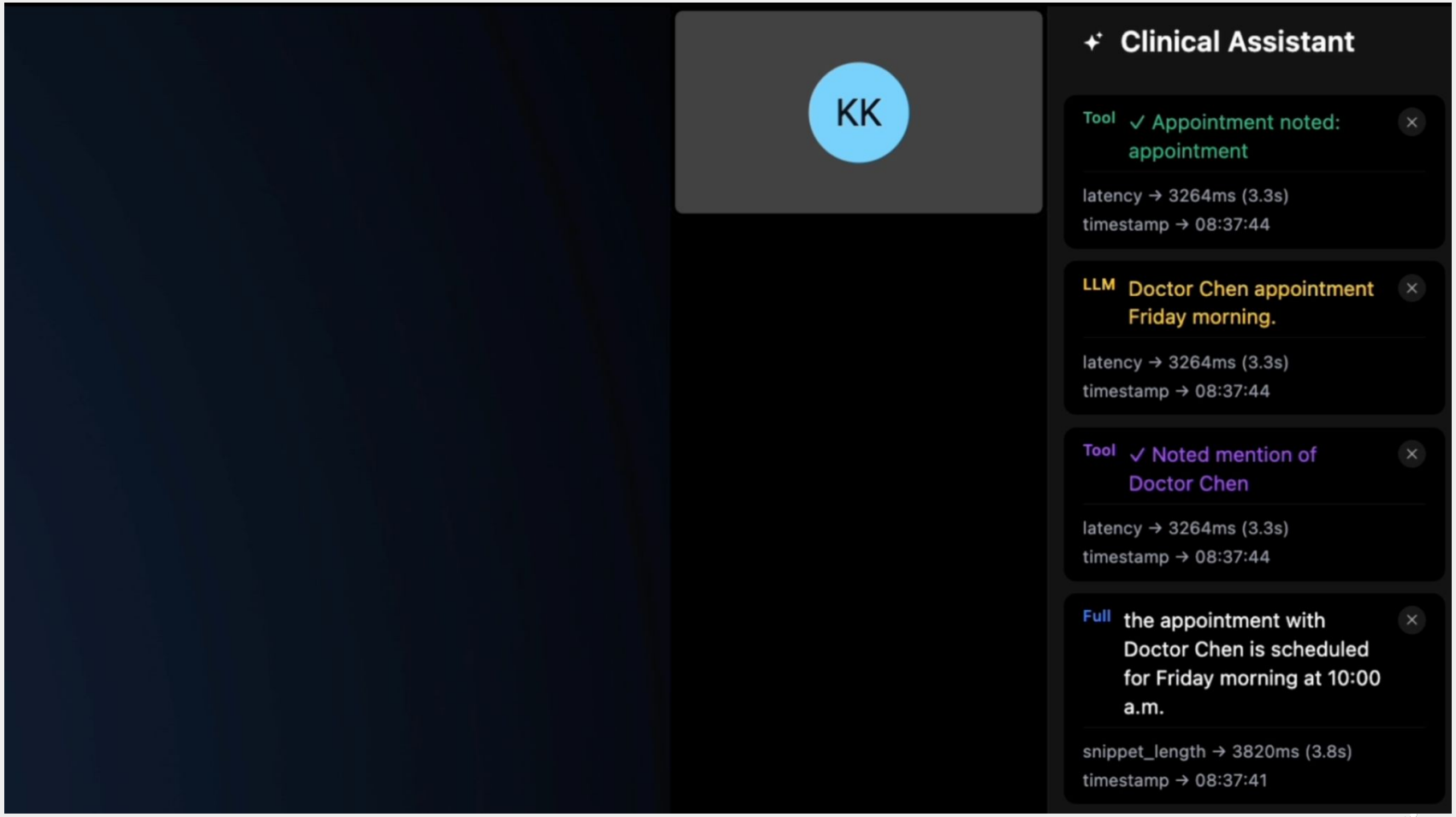
SimplePractice

LLMs can reason about actions

SimplePractice

Tools +
Reasoning →
Loop = Agent

SimplePractice



SimplePractice

SimplePractice

SimplePractice

Initial setup Bash

```bash
# Install dependencies
gem install anthropic

# Set your API key
export ANTHROPIC_API_KEY='sk-...'
```

Add initialize and definitions methods Ruby

```ruby
class Toolset
  def initialize
    @tools = build_tools
  end

  def definitions
    @tools.map { |t| t.slice(:name, :description, :input_schema) }
  end
end
```

Add execute method Ruby

```ruby
class Toolset
  # .. existing code ...

  def execute(name, input)
    tool = @tools.find { |t| t[:name] == name }
    input = JSON.parse(JSON.generate(input))
    tool[:handler].call(input)
  end
end
```

Tool Anatomy Ruby

```ruby
{
  name: 'read_file',
  description: 'Read contents of a file',
  input_schema: {
    type: 'object',
    properties: { path: { type: 'string' } },
    required: ['path']
  },
  handler: ->(input) { File.read(input['path']) }
}
```

🔗 tool use with claude

SimplePractice

Tools available to Claude

Claude Code has access to a set of powerful tools that help it understand and modify your codebase:

Tool	Description	Permission Required
Bash	Executes shell commands in your environment	Yes
Edit	Makes targeted edits to specific files	Yes
Glob	Finds files based on pattern matching	No
Grep	Searches for patterns in file contents	No
MultiEdit	Performs multiple edits on a single file atomically	Yes
NotebookEdit	Modifies Jupyter notebook cells	Yes
NotebookRead	Reads and displays Jupyter notebook contents	No
Read	Reads the contents of files	No
SlashCommand	Runs a **custom slash command**	Yes
Task	Runs a sub-agent to handle complex, multi-step tasks	No
TodoWrite	Creates and manages structured task lists	No
WebFetch	Fetches content from a specified URL	Yes
WebSearch	Performs web searches with domain filtering	Yes
Write	Creates or overwrites files	Yes

SimplePractice

Tool use examples

Here are a few code examples demonstrating various tool use patterns and techniques. For brevity's sake, the tools are simple tools, and the tool descriptions are shorter than would be ideal to ensure best performance.

- Single tool example
- Parallel tool use
- Multiple tool example
- Missing information
- Sequential tools
- Chain of thought tool use
- JSON mode

SimplePractice

Add read_file tool

Ruby

```ruby
def build_tools
  [
    {
      name: 'read_file',
      description: 'Read contents of a file',
      input_schema: {
        type: 'object',
        properties: { path: { type: 'string', description: 'File path to read' } },
        required: ['path']
      },
      handler: ->(input) { File.read(input['path']) }
    }
  ]
end
```

Add write_file tool — Ruby

```ruby
def build_tools
  [
    # ... read_file ...
    {
      name: 'write_file',
      description: 'Write content to a file',
      input_schema: {
        type: 'object',
        properties: {
          path: { type: 'string', description: 'File path to write' },
          content: { type: 'string', description: 'Content to write' }
        },
        required: ['path', 'content']
      },
      handler: ->(input) do
        File.write(input['path'], input['content'])
        "wrote #{input['path']} (#{input['content'].bytesize} bytes)"
      end
    }
  ]
end
```

SimplePractice

Add list_files tool

Ruby

```ruby
def build_tools
  [
    # ... read_file ...
    # ... write_file ...
    {
      name: 'list_files',
      description: 'List files in current directory',
      input_schema: {
        type: 'object',
        properties: { path: { type: 'string', description: 'Directory path (optional)' } }
      },
      handler: ->(input) do
        path = input['path'] || '.'
        files = Dir.entries(path).reject { |f| f.start_with?('.') }.sort
        files.join("\n")
      end
    }
  ]
end
```

tice

Add bash tool Ruby

```ruby
def build_tools
  [
    # ... read_file ...
    # ... write_file ...
    # ... list_files ...
    {
      name: 'bash',
      description: 'Execute a shell command',
      input_schema: {
        type: 'object',
        properties: { command: { type: 'string', description: 'Command to run' } },
        required: ['command']
      },
      handler: ->(input) { `#{input['command']} 2>&1` }
    }
  ]
end
```

More tools ≠
better agent

Add initialize method

Ruby

```ruby
class Agent
  def initialize
    @client = Anthropic::Client.new(api_key: ENV.fetch('ANTHROPIC_API_KEY'))
    @toolset = Toolset.new
    @messages = []
  end
end
```

Add run method Ruby

```ruby
class Agent
  # ... existing code ...

  def run
    UI.banner

    loop do
      UI.prompt
      input = gets&.strip
      break if input.nil? || input == 'exit'
      next if input.empty?

      @messages << { role: :user, content: input }
      handle_conversation
    end
  end
end
```

Messages API Protocol — Ruby

```ruby
# Turn 1: User asks
{ role: :user, content: "read config.json" }

# Turn 2: Claude responds with tool use
{ role: :assistant, content: [
  { type: :text, text: "Let me read that" },
  { type: :tool_use, id: "1", name: "read_file", input: {path: "config.json"} }
]}

# Turn 3: You respond with results
{ role: :user, content: [
  { type: :tool_result, tool_use_id: "1", content: "{...}" }
]}
```

Start handle_conversation method Ruby

```ruby
class Agent
  # ... existing code ...

  def handle_conversation
    loop do
      response = @client.messages.create(
        model: 'claude-sonnet-4-5',
        max_tokens: 1024,
        messages: @messages,
        tools: @toolset.definitions
      )

      text_blocks = response.content.select { |b| b.type == :text }
      tool_uses = response.content.select { |b| b.type == :tool_use }
    end
  end
end
```

Display response and check for tools Ruby

```ruby
class Agent
  # ... existing code ...

  def handle_conversation
    loop do
      # ... existing code ...
      if text_blocks.any?
        text = text_blocks.map(&:text).join
        UI.agent(text)
      end

      if tool_uses.empty?
        @messages << { role: :assistant, content: text_blocks.map(&:text).join }
        break
      end
    end
  end
end
```

Execute tools

Ruby

```ruby
class Agent
  # ... existing code ...

  def handle_conversation
    loop do
      # ... existing code ...
      tool_results = tool_uses.map do |tool_use|
        UI.tool_call(tool_use.name)
        result = @toolset.execute(tool_use.name, tool_use.input)
        UI.tool_result(result)

        { type: :tool_result, tool_use_id: tool_use.id, content: result }
      end
    end
  end
end
```

Feed results back to Claude

Ruby

```ruby
class Agent
  # ... existing code ...

  def handle_conversation
    loop do
      # ... existing code ...
      @messages << {
        role: :assistant,
        content: [
          *text_blocks.map { |b| { type: :text, text: b.text } },
          *tool_uses.map { |tu| { type: :tool_use, id: tu.id, name: tu.name, input: tu.input }
}
        ]
      }

      @messages << { role: :user, content: tool_results }
    end
  end
end
```

Take it for a spin!

```
$ ruby agent.rb
```