The Pelita contest

(a brief introduction)

In short

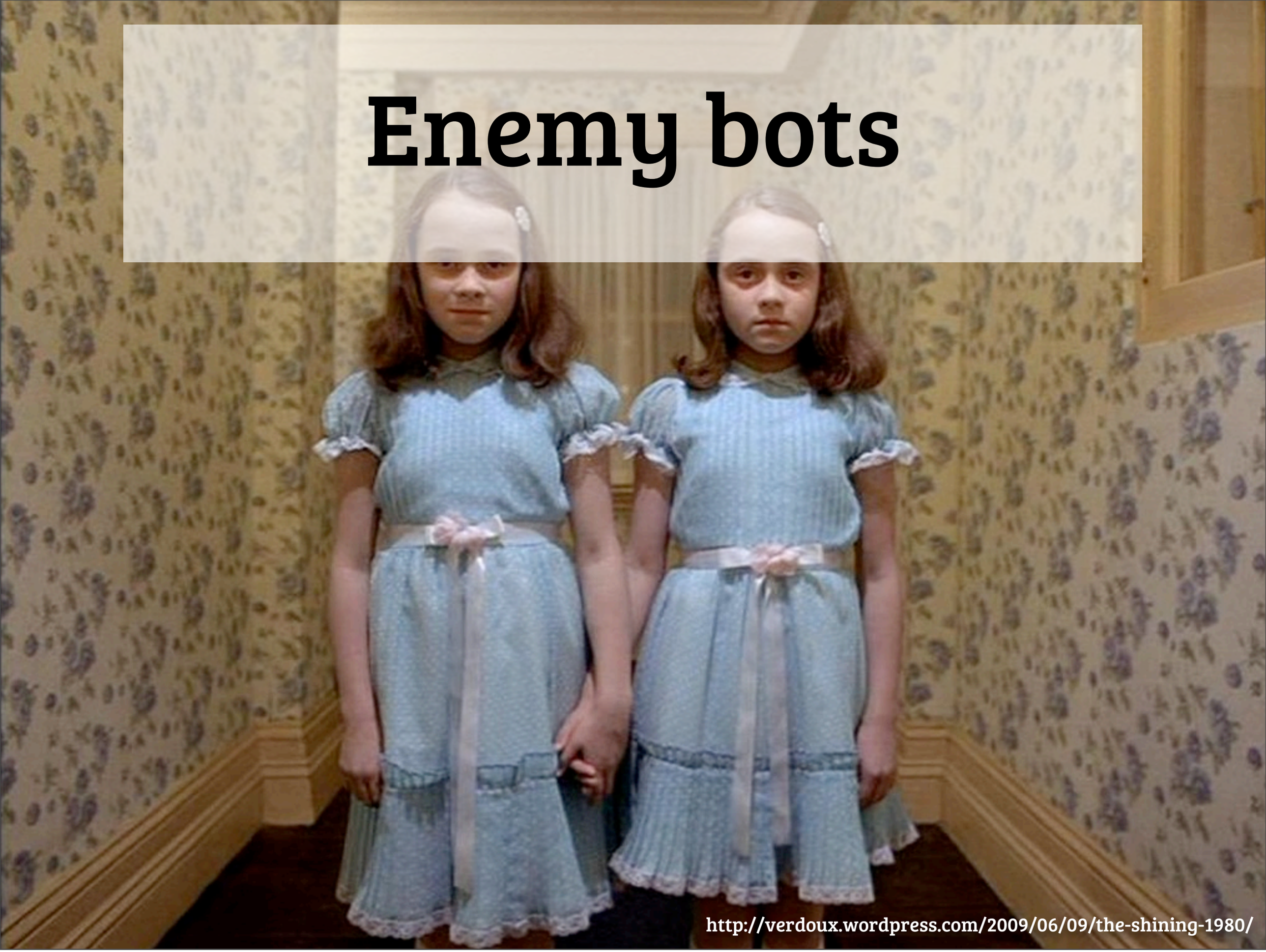
A maze

http://verdoux.wordpress.com/2009/06/09/the-shining-1980/

Moving around

http://verdoux.wordpress.com/2009/06/09/the-shining-1980/

Enemy bots

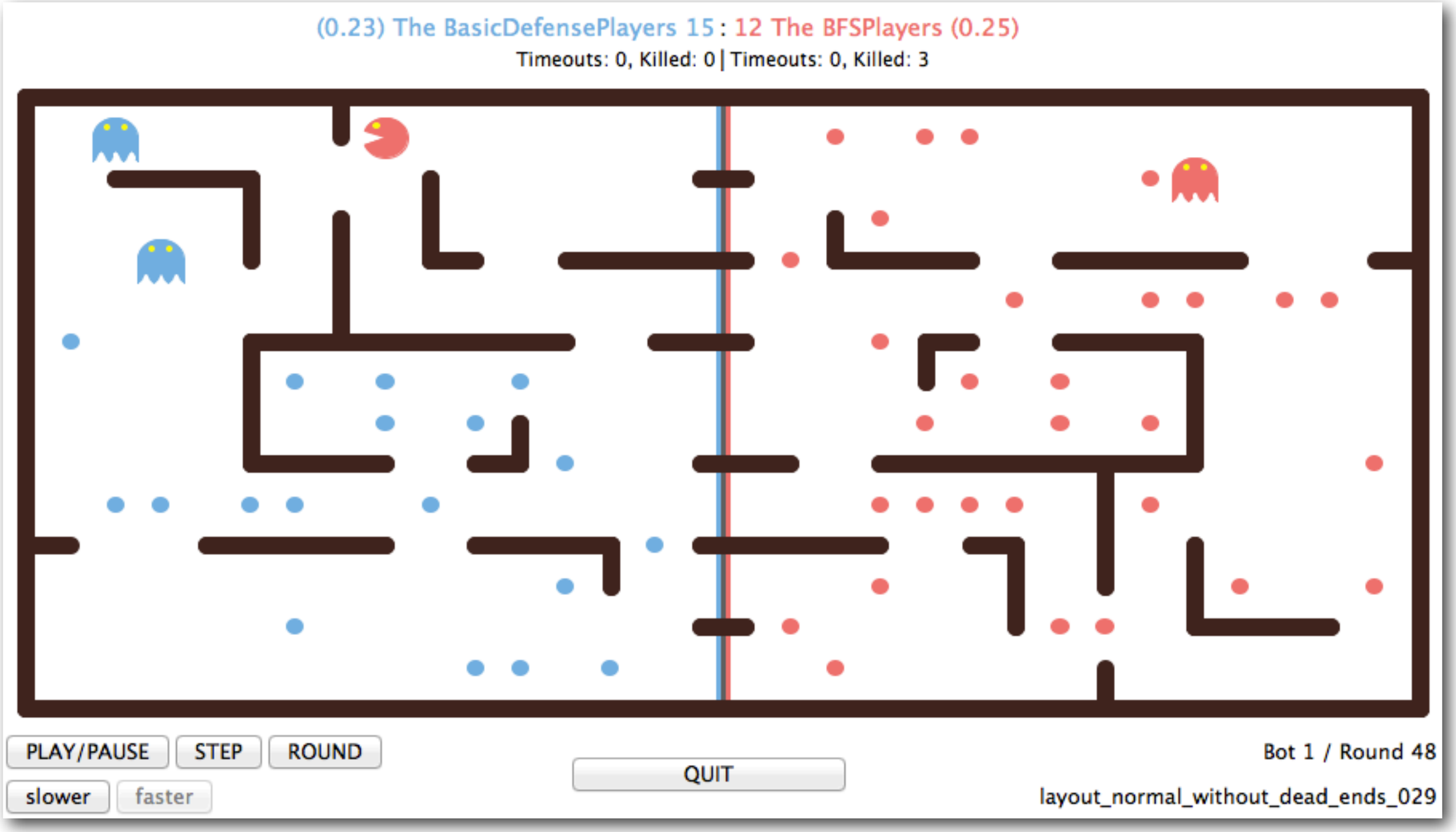


http://verdoux.wordpress.com/2009/06/09/the-shining-1980/

Attack

http://img.timeinc.net/time/photoessays/2008/top10_1950s/top10_1950s_them.jpg

Pelita


(0.23) The BasicDefensePlayers 15 : 12 The BFSPlayers (0.25)
Timeouts: 0, Killed: 0 | Timeouts: 0, Killed: 3
PLAY/PAUSE
STEP
ROUND
slower
faster
QUIT
Bot 1 / Round 48
layout_normal_without_dead_ends_029

Before you ask

- Pelita
- Actor-based Toolkit for Interactive Language Education in Python

- 'Pill-eater'
- Created 2011–2012 especially for the summer school
- (Idea from John DeNero and Dan Klein, UC Berkeley[1])

[1] http://www.denero.org/content/pubs/eaai10_denero_pacman.pdf

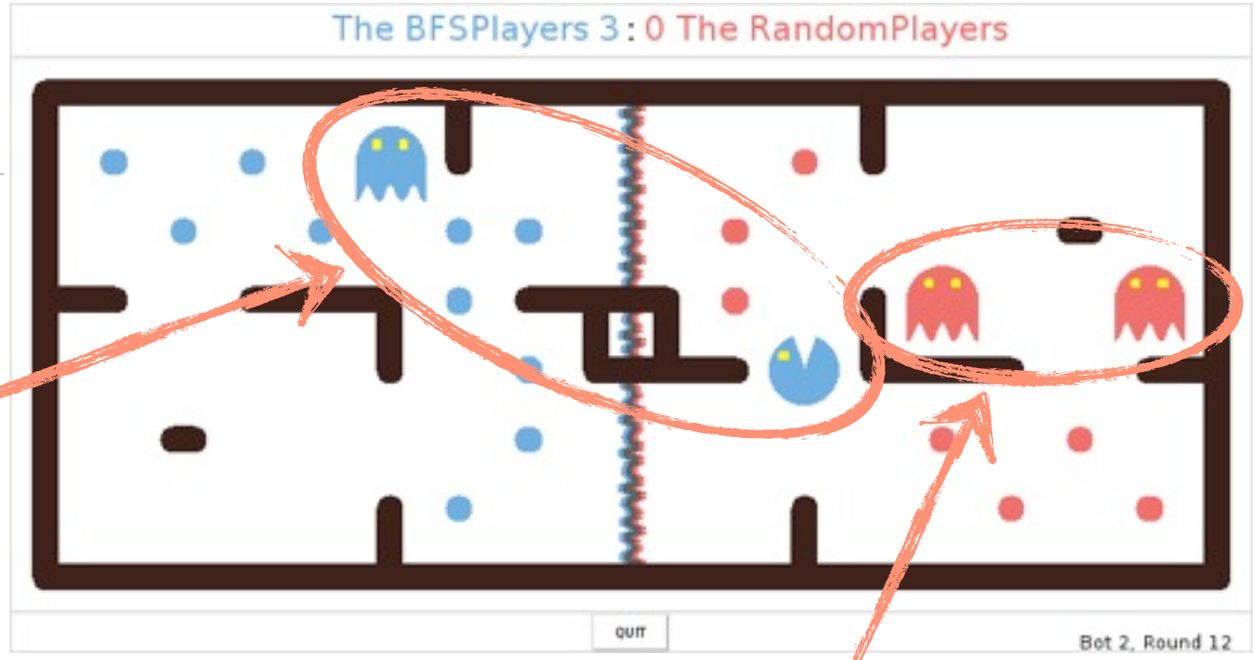
git describe && git shortlog -sn | cut -f2

- v0.2.0-rc1

Rike-Benjamin Schuppner
Valentin Haenel
Tiziano Zito
Zbigniew Jędrzejewski-Szmek
Bastian Venthur
Pietro Berkes
Pauli Virtanen
Nicola Chiapolini
Ola Pidde
Sasza Kijek
Anna Chabuda
Francesc Alted
Zbigniew Jędrzejewski-Szmek
Christian Steigies

Overview

- Each Team owns two Bots

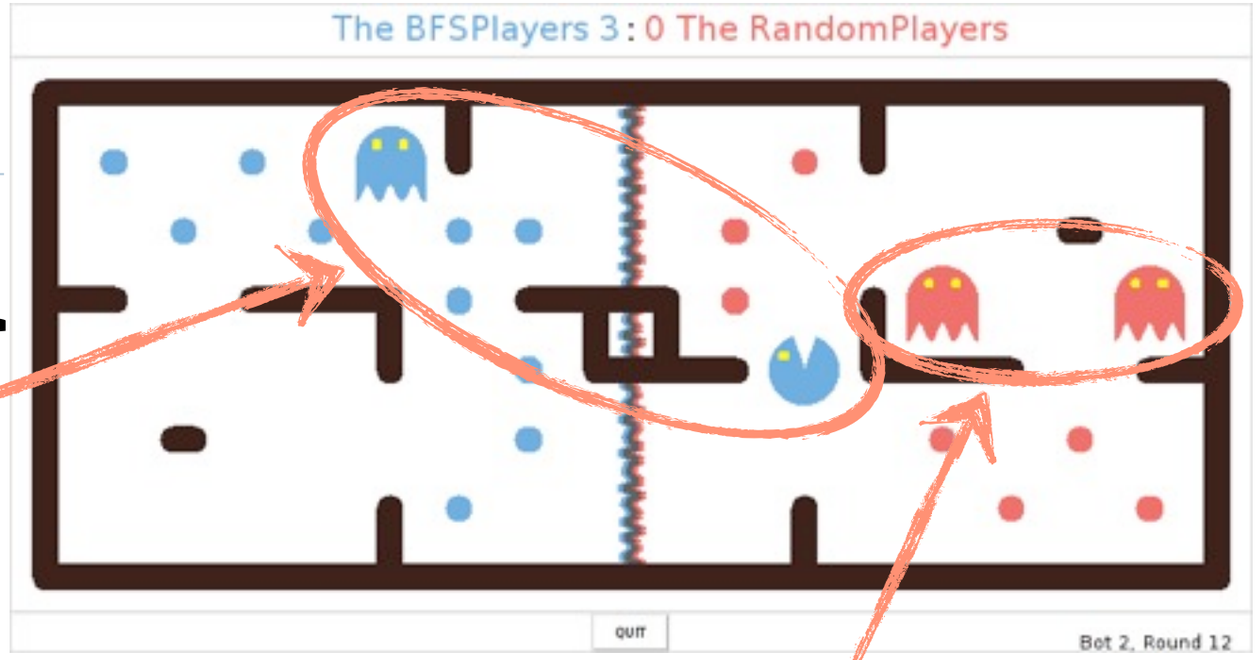


Bots for team 0

Bots for team 1

Overview

- Each Team owns two Bots
- Each Bot is controlled by a Player



Bots for team 0

Player for team 0

Bots for team 1

Player for team 1

```
from pelita.datamodel import east
from pelita.player import AbstractPlayer

class UnidirectionalPlayer(AbstractPlayer):
    def get_move(self):
        return east
```

```
from pelita.datamodel import west
from pelita.player import AbstractPlayer

class UnidirectionalPlayer(AbstractPlayer):
    def get_move(self):
        return west
```

Overview

- Each Team owns two Bots
- Each Bot is controlled by a Player
- Harvester or Destroyer Bots

Overview

- Each Team owns two Bots
- Each Bot is controlled by a Player
- Harvester or Destroyer Bots
- Bots are Destroyers in homezone
- Harvesters in enemy's homezone
- Game ends when all food pellets are eaten

The rules

- Eating: When a Bot eats a food pellet, the food is permanently removed and one point is scored for that Bot's team.

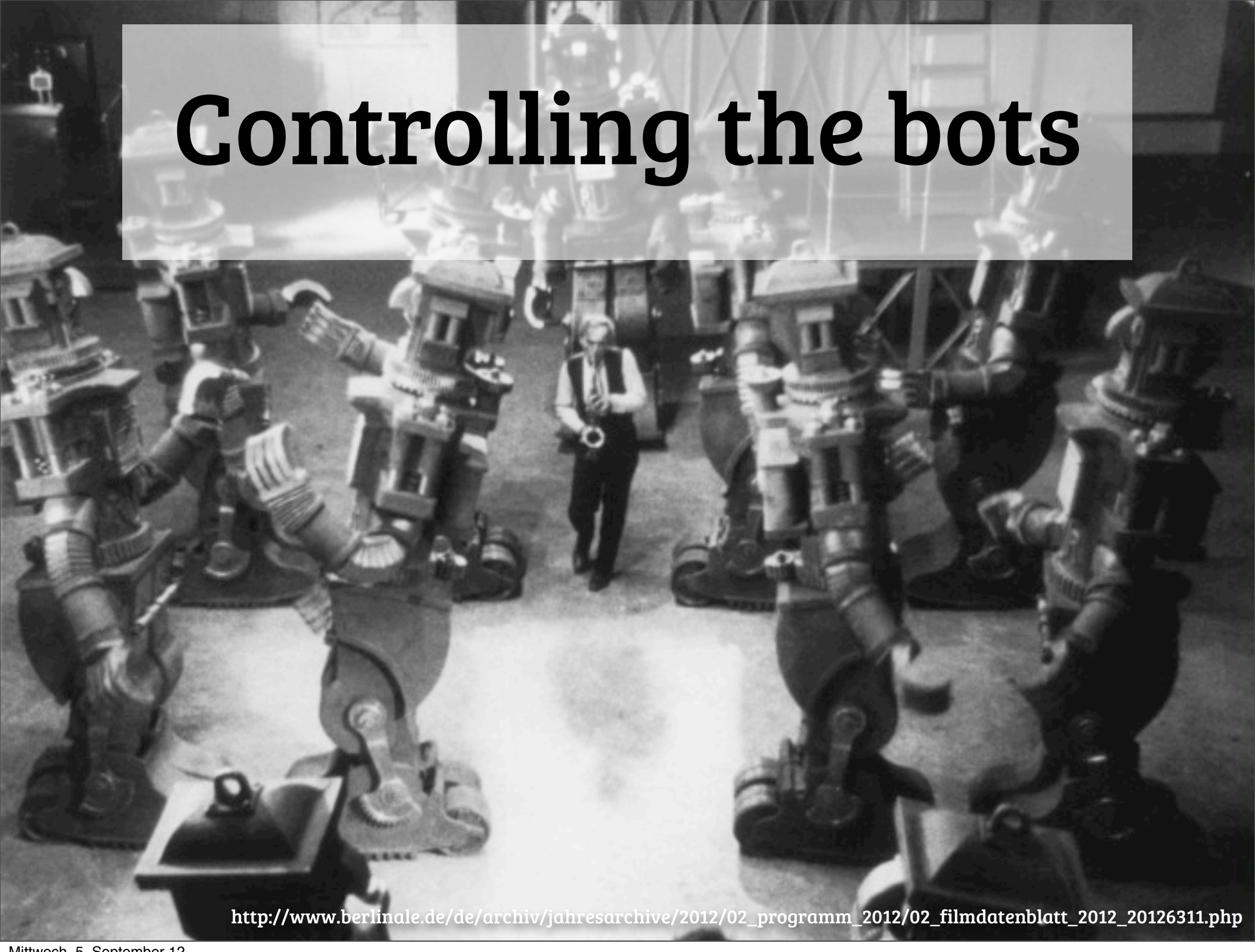
- Timeout: Each Player only has 3 seconds to return a valid move. If it doesn't, a random move is executed. (All later return values are discarded.)
  5 timeouts and you're out!

- Eating another Bot: When a Bot is eaten by an opposing destroyer, it returns to its starting position (as a harvester). 5 points are awarded for eating an opponent.

- Winning: A game ends when either one team eats all of the opponents' food pellets, or the team with more points after 300 rounds.

- Observations: Bots can only observe an opponent's exact position, if they or their teammate are within 5 squares of the opponent bot. If they are further away, the opponent's positions are noised.

Controlling the bots



http://www.berlinale.de/de/archiv/jahresarchive/2012/02_programm_2012/02_filmdatenblatt_2012_20126311.php

Short how-to

- **Subclass AbstractPlayer:**

```
class MyPlayer(AbstractPlayer):
  def get_move():
    return stop
```

- **Build a SimpleTeam:**

```
our_team = SimpleTeam("yeeeh", MyPlayer(), MyPlayer())
```

- **Export:**

```
def factory(): return our_team
```

My first players

Pelita imports

Inherit from AbstractPlayer

Use the Player API

```
from pelita.datamodel import east
from pelita.player import AbstractPlayer

class UnidirectionalPlayer(AbstractPlayer):
    def get_move(self):
        return east

class DrunkPlayer(AbstractPlayer):
    def get_move(self):
        directions = self.legal_moves
        random_dir = self.rnd.choice(directions)
        return random_dir
```

- Careful: Invalid return values of get_move result in a random move.

API examples

- In your get_move method, information about the current universe and food situation is available. See the documentation for more details.

- self.current_pos
  Where am I?

- self. me
  Which bot am I controlling?

- self. enemy_bots
  Who and where are the other bots?

- self. enemy_food
  Which are the positions of the food pellets?

- self. current_uni
  Retrieve the universe you live in.

- self. current_uni.maze
  How does my world look like?

- self. legal_moves
  Where can I go?

Testing

http://verdoux.wordpress.com/2009/06/09/the-shining-1980/

Testing

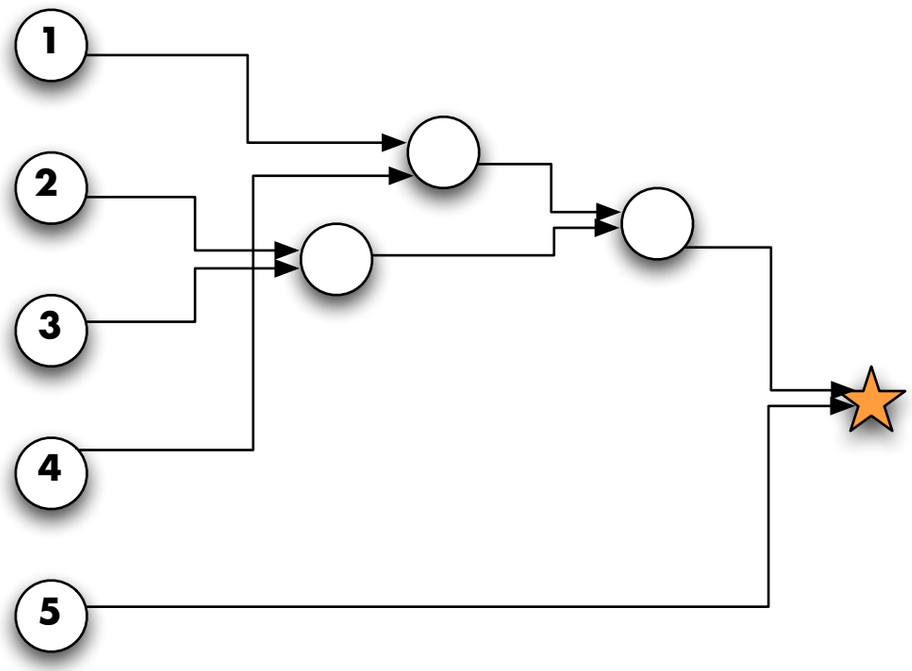
- Two ways to test your Players
- first: Simply run the game and test by watching
  - $ ./pelitagame MyTeam EnemyTeam
- second: Write unittests and test by testing
  - Example in the template

Tournament

http://magiaeimagem.files.wordpress.com/2010/02/ingmar-bergman-the-seventh-seal.jpg

Tournament mode

- Two parts
  - first: all-against-all
  - then: knockout
  - bonus:
    tutor-humiliation round



Tournament setup

- Group repository:
  git clone <name>@python.g-node.de/git/groupN
- Make it a module. (Add __init__.py)
- Export ‘factory’ method:

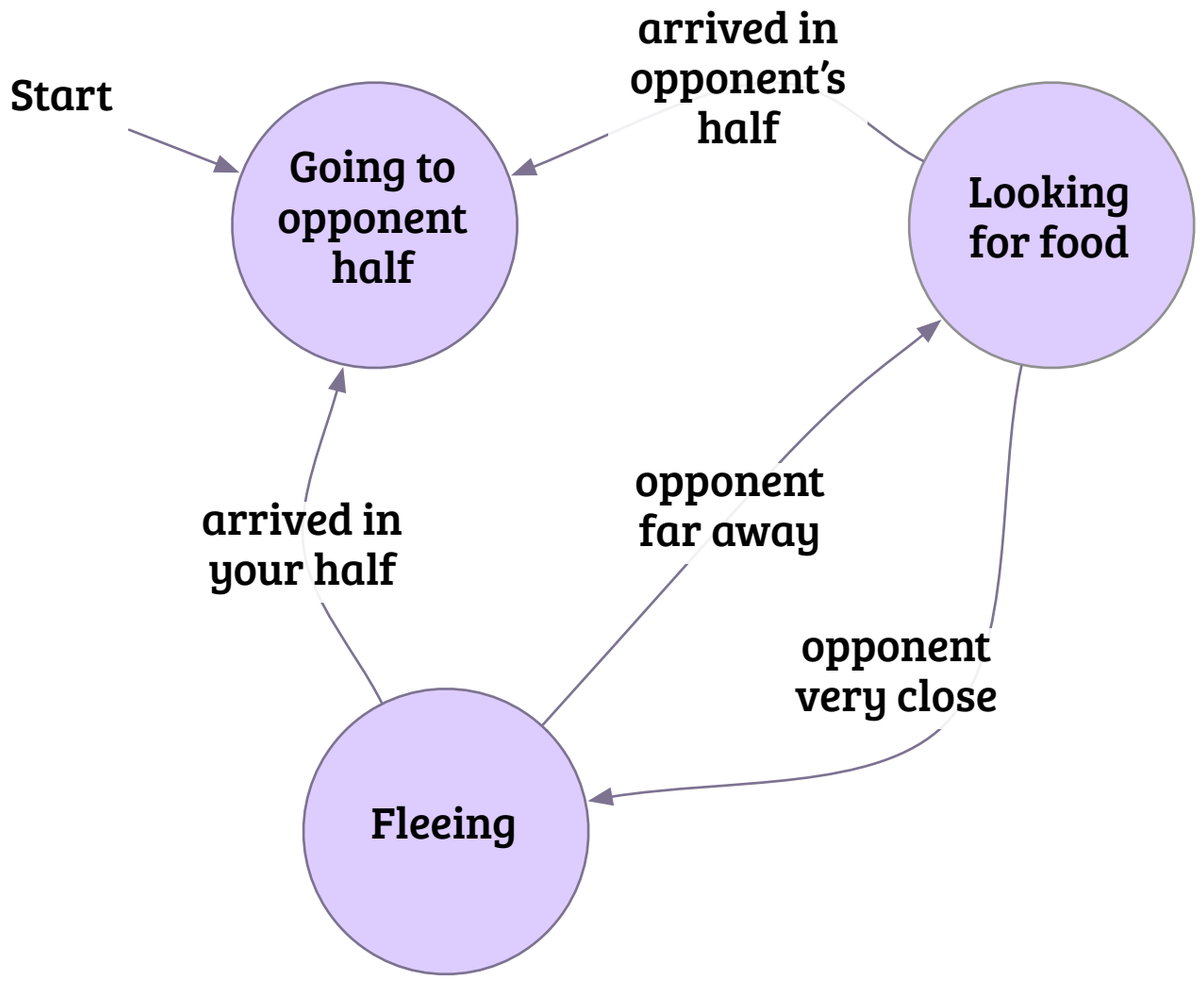
```
def factory():
    return SimpleTeam("The Winners", MyPlayer(), MyPlayer())
```

- Template and more information to be found in the wiki and in the documentation

Strategies

Finite state machine

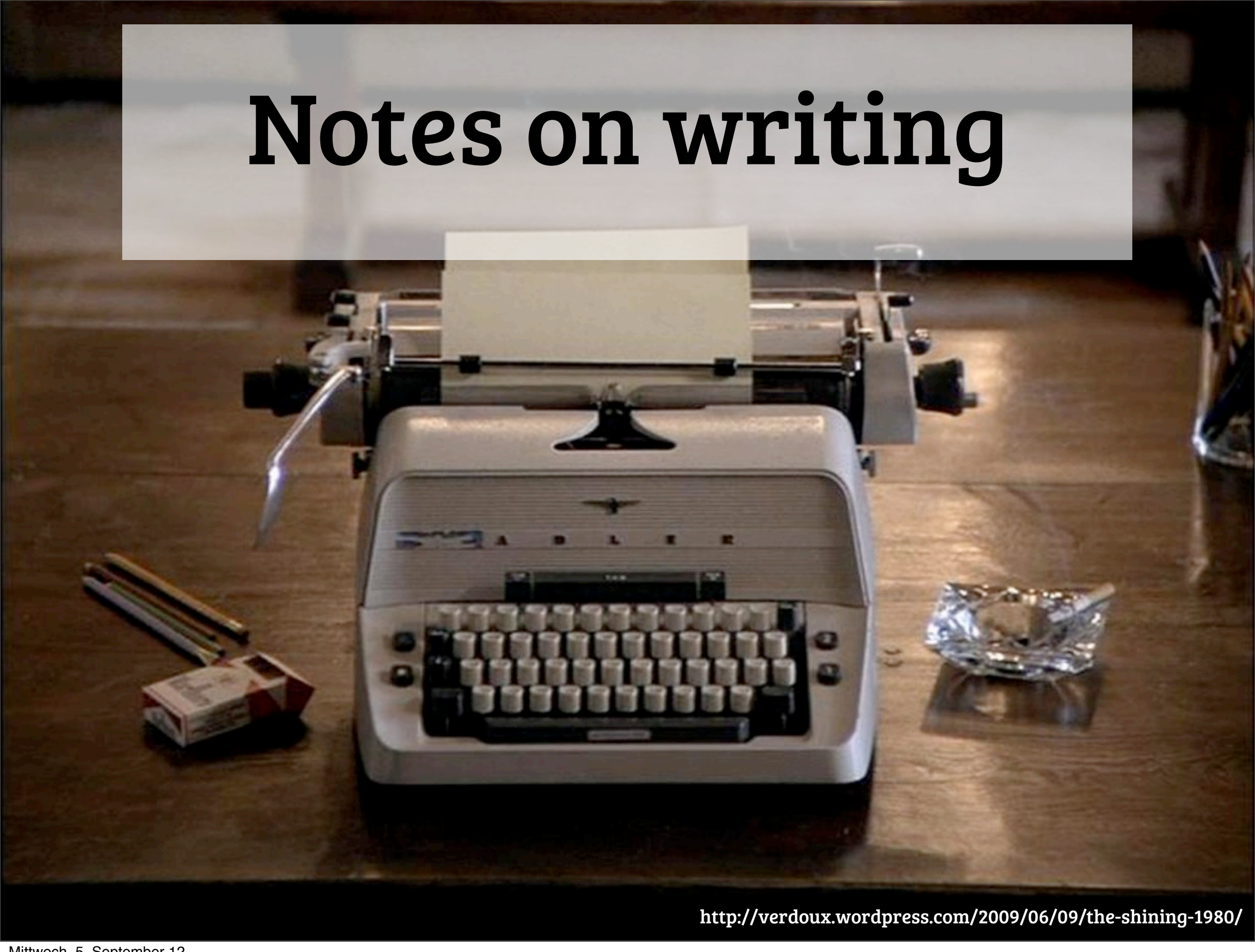
- Evaluate the current situation and choose the algorithm accordingly



Look-ahead Player

- Create a function which calculates a score for each situation, eg.
  - value(game_state) = −1 × distance_from_nearest_food + 100 × score
- At each turn do
  - get the legal moves for your bot
  - request the future universe, given one of the actions
    self.current_uni.copy().move_bot(self._index, direction)
  - compute the score
  - choose the direction with the best score

Notes on writing



http://verdoux.wordpress.com/2009/06/09/the-shining-1980/

Notes on writing

- Mazes won't have dead-ends
- Hard to catch another bot which outruns you
- We'd like to see bots which combine their powers and attack from two sides

Notes on writing

- Think about shortest-path algorithms
- Keep track of opponents
- Investigate communication between the Players
- Re-use your code
- Think about working in a team

Notes on writing

- Use the internal rng:
- instead of
  - random.choice
- you use
  - self.rnd.choice
- (more stable)

Getting ready

- **Clone the repo:**
  git clone git://github.com/ASPP/pelita.git
- **Run a simple demo game:**
  ~/pelita/pelitagame
- **For help:**
  ~/pelita/pelitagame --help
- **See the Pelita documentation:**
  http://ASPP.github.com/pelita
- **Questions? Ask us.**
- **Write your own player already!**