

The **Pelita** contest

(a brief introduction)

Advanced Scientific Programming in Python

#aspp2015

The **Pelita** contest

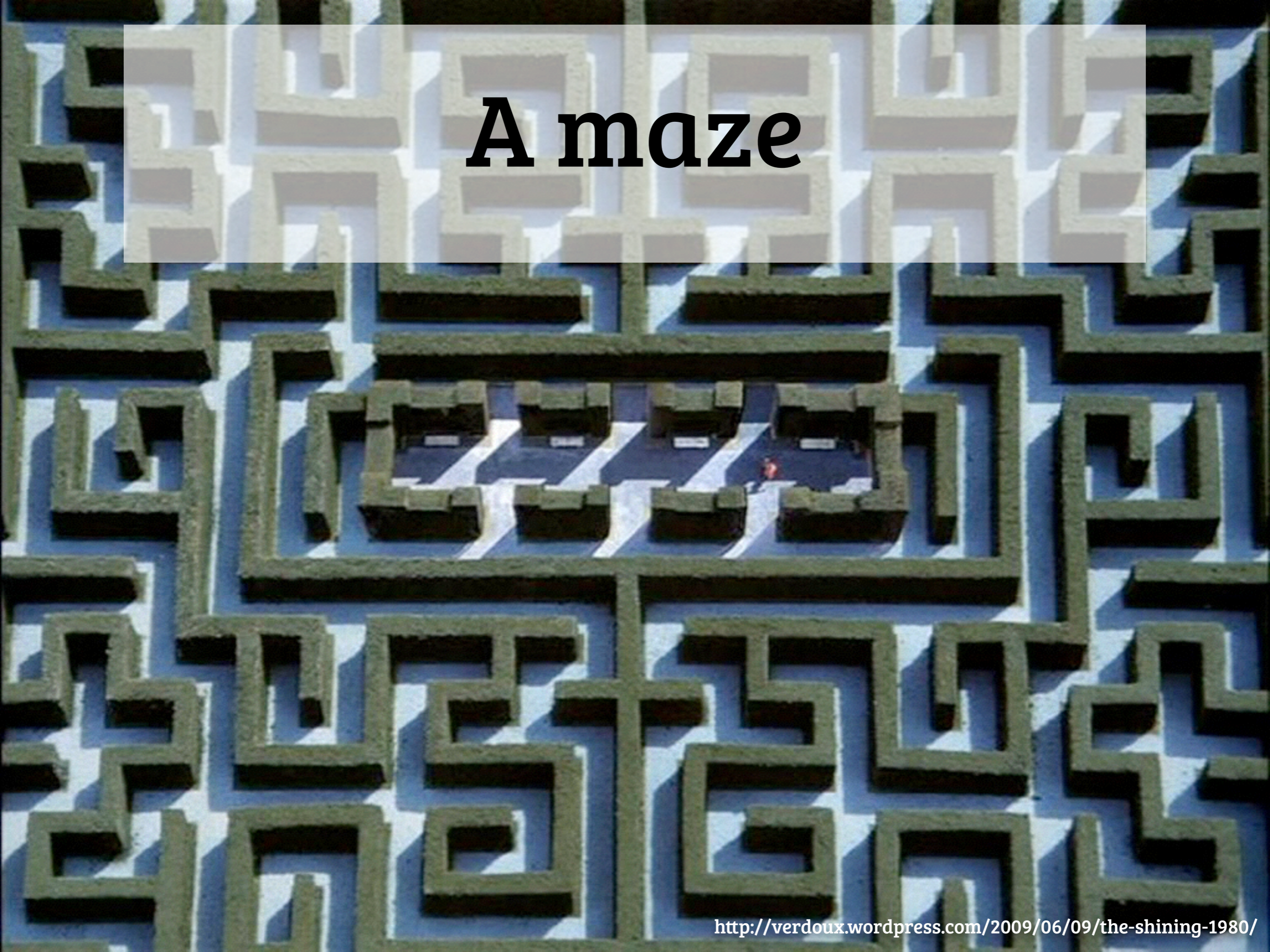
(a brief introduction)

Rike-Benjamin Schuppner
Institute for Theoretical Biologie | HU Berlin

rikebs@debilski.de // debilski.de // [@debilski](https://twitter.com/debilski)

In short

A maze



Moving around



Enemy bots



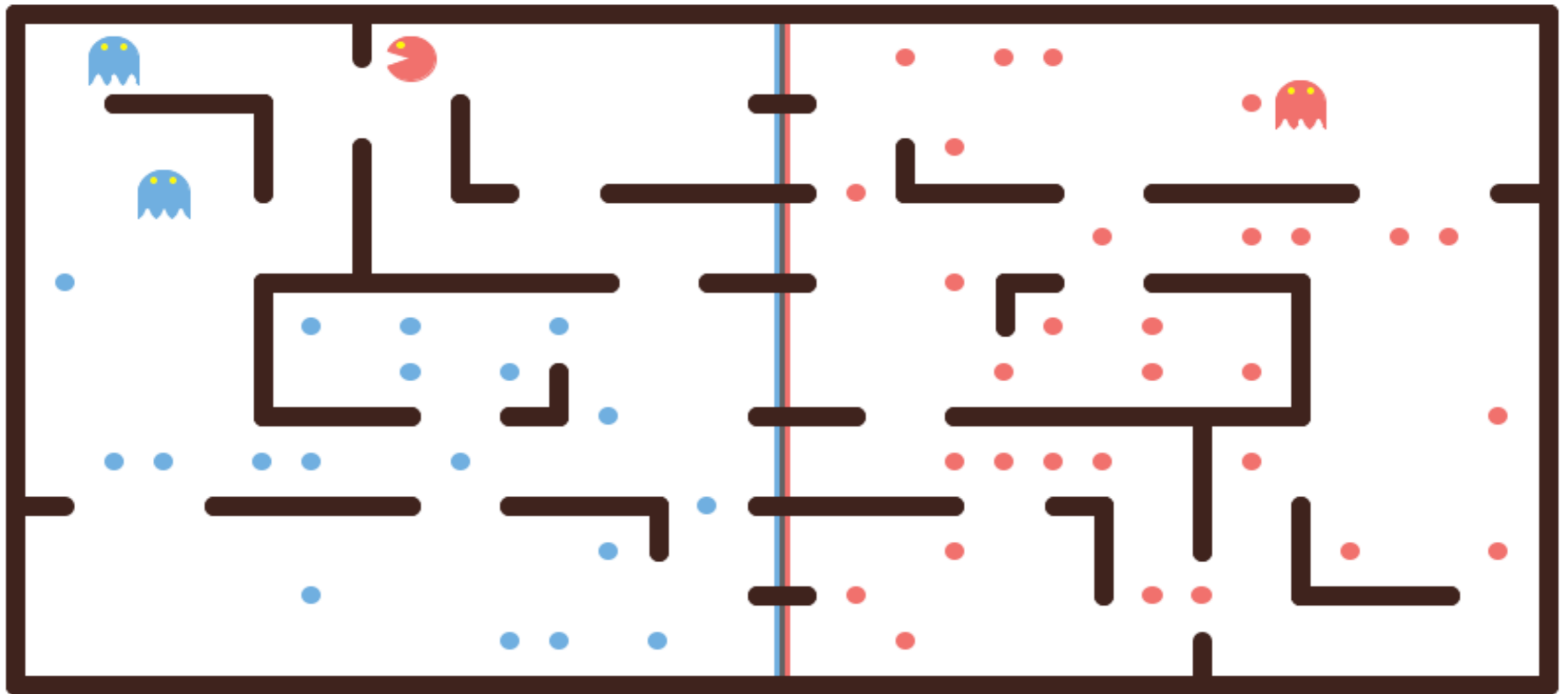
Attack



Pelita

(0.23) The BasicDefensePlayers 15 : 12 The BFSPlayers (0.25)

Timeouts: 0, Killed: 0 | Timeouts: 0, Killed: 3



PLAY/PAUSE

STEP

ROUND

QUIT

slower

faster

Bot 1 / Round 48

layout_normal_without_dead_ends_029

Before you ask

- **Pelita**
- **A**ctor-based **T**oolkit for **I**nteractive **L**anguage **E**ducation in **P**ython
- 'Pill-eater'
- Created 2011–2012 especially for the summer school
- (Idea from John DeNero and Dan Klein, UC Berkeley¹)

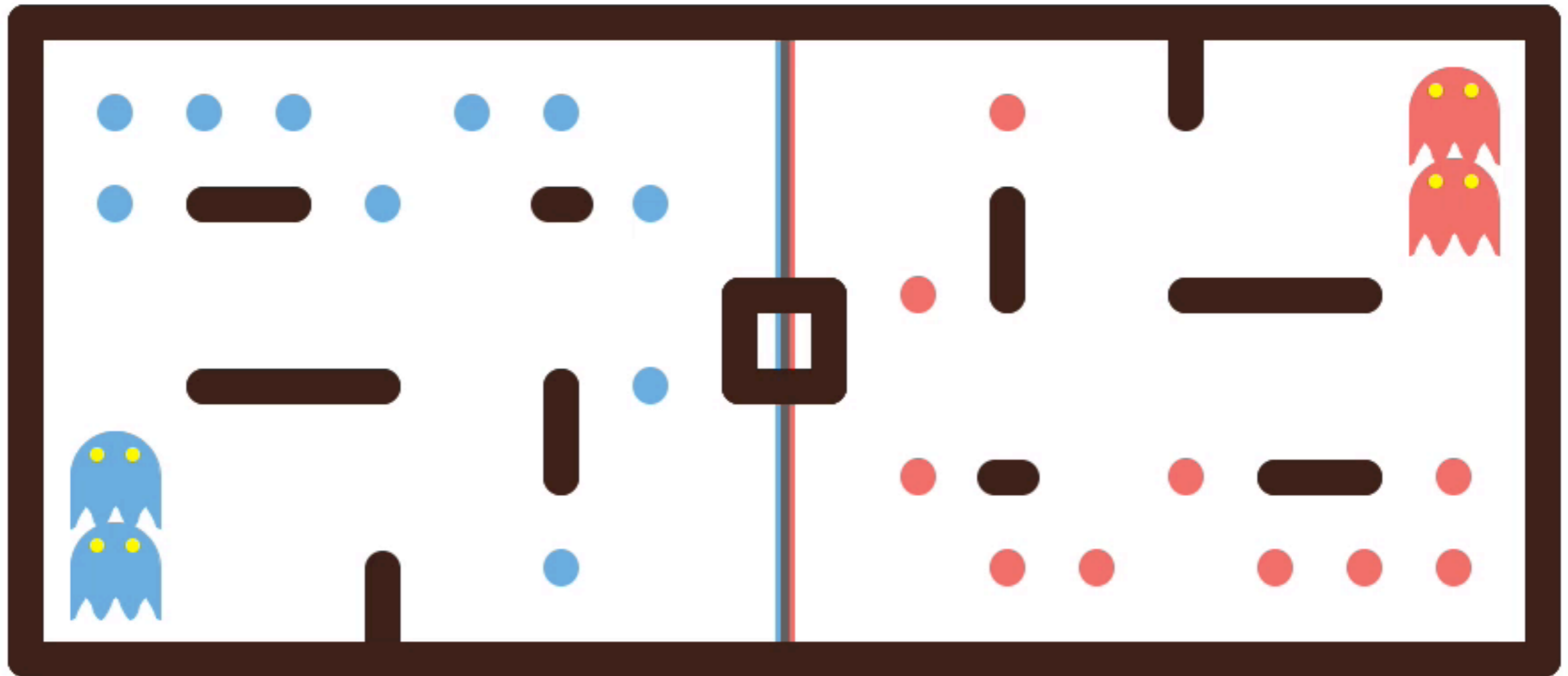
¹ http://www.denero.org/content/pubs/eaai10_denero_pacman.pdf

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

- **v0.2.0-272-g1f2740f**
Rike-Benjamin Schuppner
Valentin Haenel
Tiziano Zito
Zbigniew Jędrzejewski-Szmek
Bastian Venthur
Pietro Berkes
Nicola Chiapolini
Pauli Virtanen
abject
Anna Chabuda
Jakob Jordan
Sasza Kijek
Francesc Alted
Christian Steigies
Zbigniew Jędrzejewski-Szmek
Bartosz Telenczuk
Ola Pidde

Overview

(0.00) The FoodEatingPlayers 0 : 0 The RandomExplorerPlayers (0.00)



PLAY/PAUSE STEP ROUND
slower faster show grid

QUIT

Overview

- ▶ Each Team owns two Bots

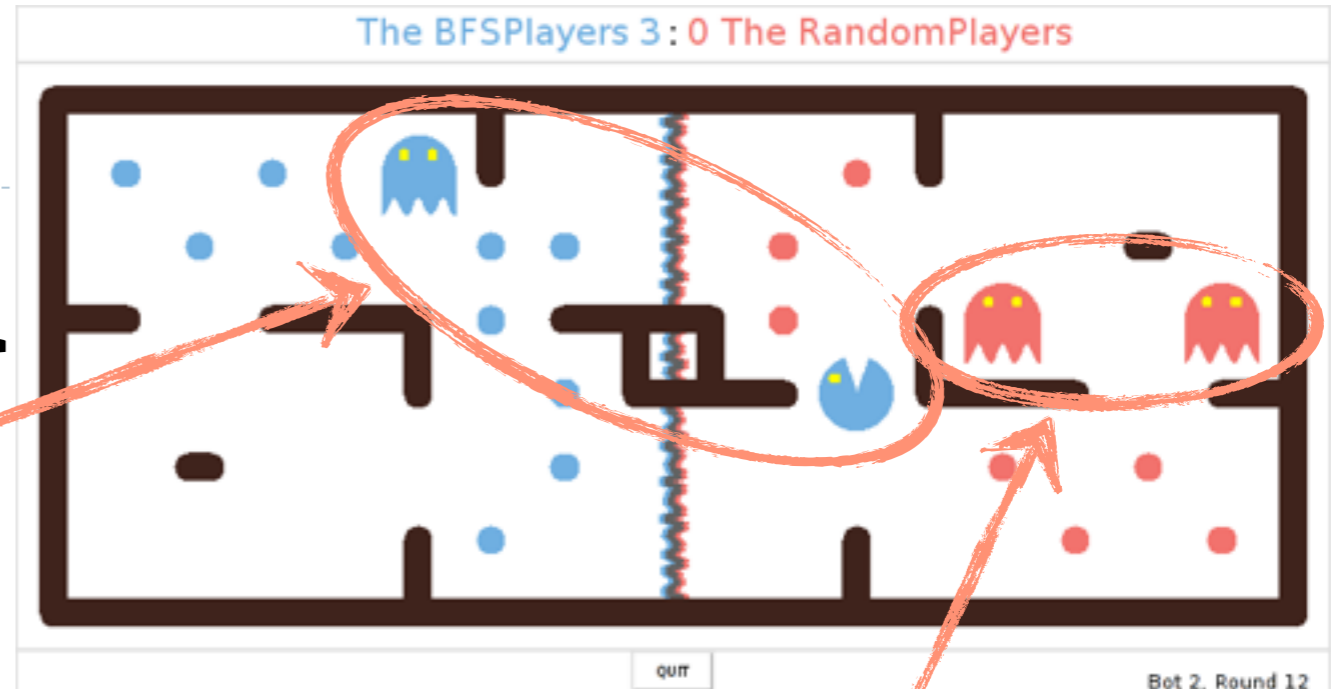
Bots for team 0



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

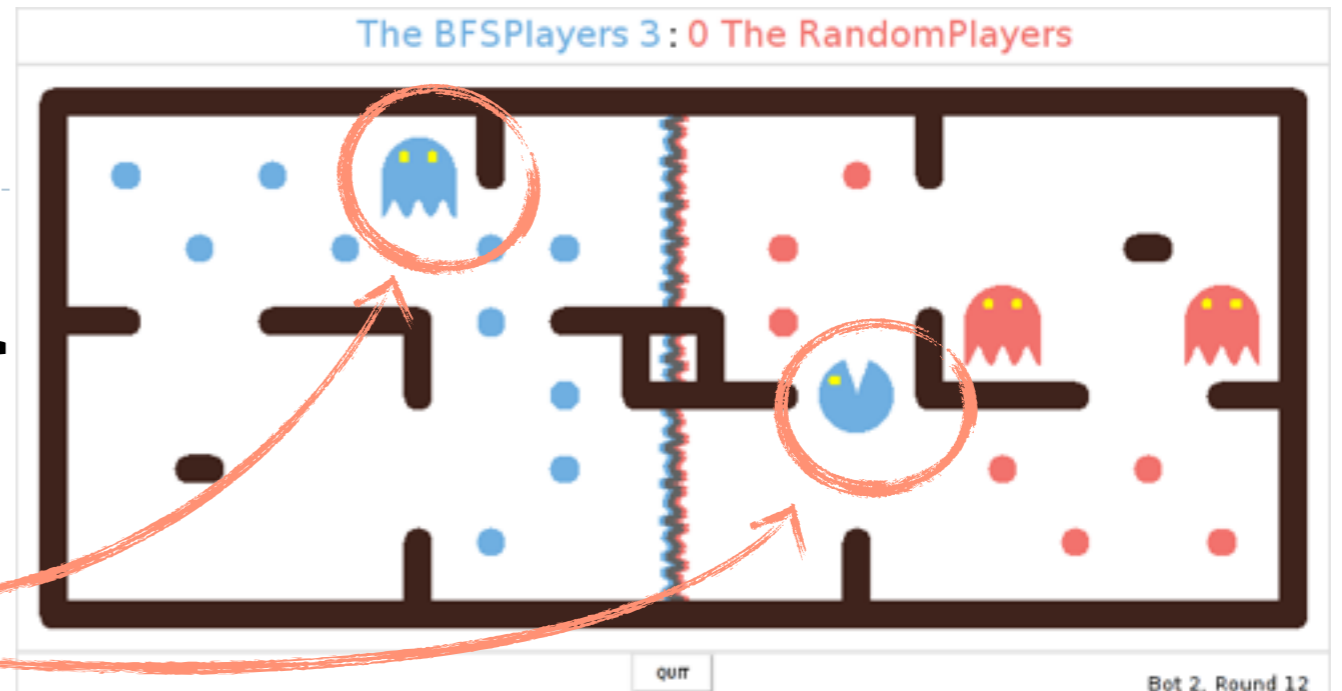
class UnidirectionalPlayer(AbstractPlayer):
    def get_move(self):
        return east
```

```
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



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



My first players

Pelita imports

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

Inherit from
AbstractPlayer

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

Use the Player
API

```
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?
- `self.me.is_destroyer`
Am I dangerous?

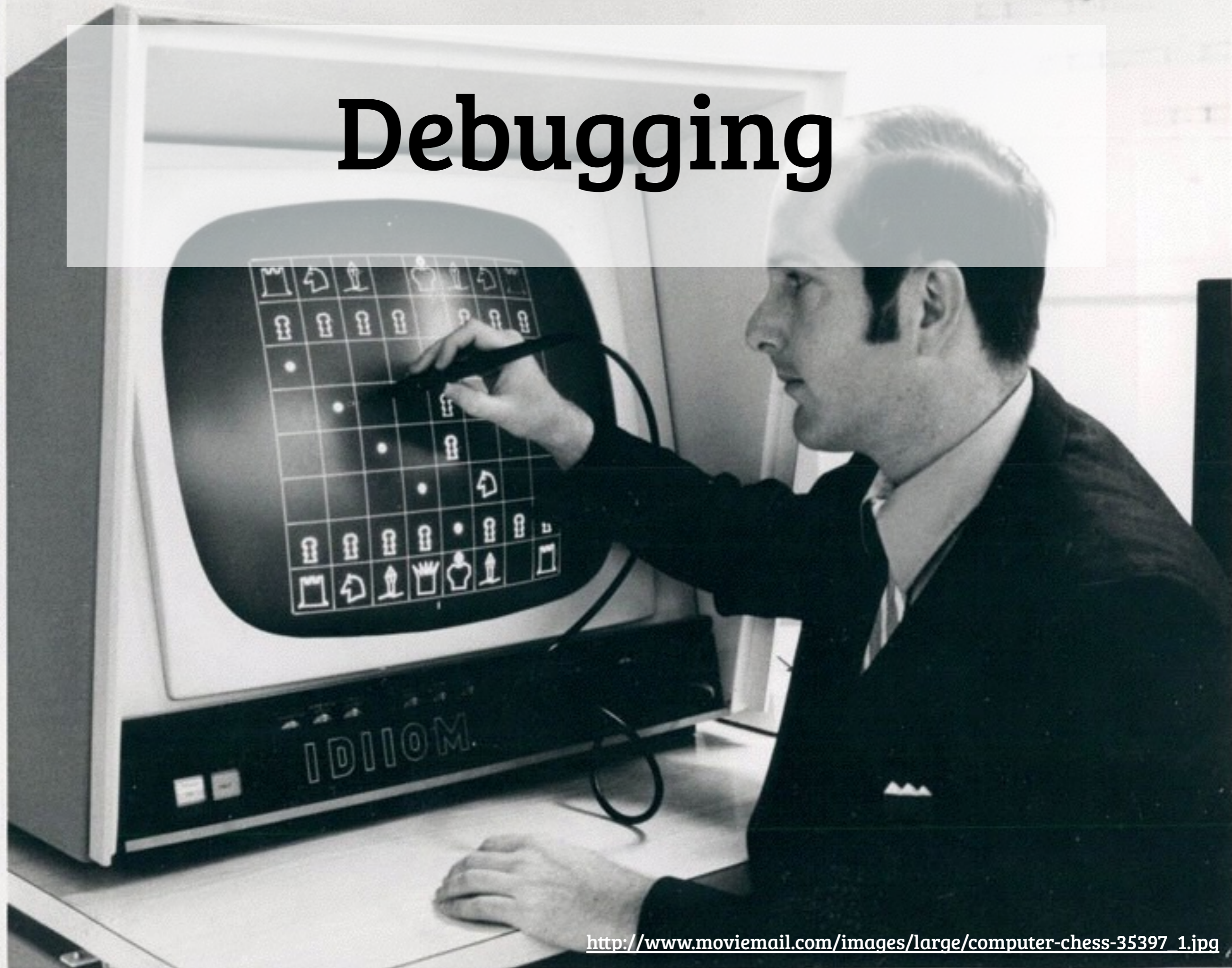
Building a team

- A team consists of two players (and a name)
- Create it using the **SimpleTeam** class
 - `SimpleTeam("Magnificent Team", GoodPlayer(), RemarkablePlayer())`
- Export your team using the **factory** function
 - `def factory():
 return SimpleTeam(...)`

Demo bots

- **In ./players directory**
- **There are hidden bots on our servers**
 - **We tell you how to use them when it's time**
- **Also depends on the network**

Debugging



Debugging

- Use a pre-defined DebuggablePlayer to explore the API
- `class DebuggablePlayer(AbstractPlayer):`
 `def get_move(self):`
 `direction = datamodel.stop`
 `pdb.set_trace()`
 `return direction`
- `./pelitagame --no-timeout DebuggablePlayer`
- (Pdb) `p self.me`

Testing



Testing

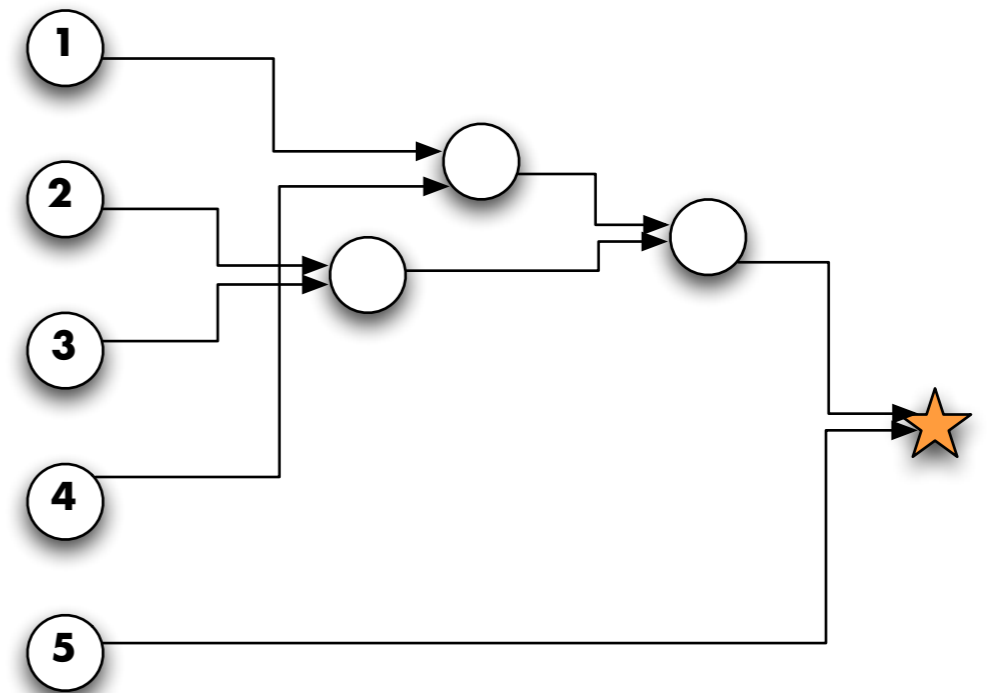
- **Two ways to test your Players**
- **first: Simply run the game and test by watching**
 - **\$./pelitagame MyTeam EnemyTeam**
- **second: Write unittests and test by testing**
 - **Example in the template**

Tournament



Tournament

- **Two stages mode**
 - **first: all-against-all (round robin)**
 - **then: knockout**
 - **bonus:**
tutor-humiliation round



Tournament

- Group repository:
`git clone <name>@python.g-node.org/git/groupN`
- It contains a module in `team/`. (Uses `__init__.py`)
- Exports a 'factory' method:

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

- Run it as
`./pelitagame ../groupN/team`
- Additionally contains util and testing repositories

Strategies



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



Notes on writing

- **Mazes don'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 random number generator:**
- **instead of**
 - **random.choice**
- **you use**
 - **self.rnd.choice**
- **(more stable)**

Notes on writing

- **The match environment:**
 - **numpy** is installed
 - also: **pylint** (just so you know)
 - additional packages may or may not be negotiable

Getting ready

- Clone the pelita and group repos:
`git clone git://github.com/ASPP/pelita.git`
`git clone <name>@python.g-node.org/git/groupN`
- Run a simple demo game:
`~/pelita/pelitagame groupN/team`
- For help:
`~/pelita/pelitagame --help`
- See the **Pelita** documentation:
<http://ASPP.github.io/pelita>
- Questions? Ask us.
- Vent your frustration: [#aspp2015](#)

Repo closes



Repo closes

Saturday, 5pm.

Movie stills

- **'Gibel sensatsii' (1935, dir. Aleksandr Andriyevsky)**
- **'Them' (1954, dir. Gordon Douglas)**
- **'The Ten Commandments' (1956, dir. Cecil B. DeMille)**
- **'Det sjunde inseglet' (1957, dir. Ingmar Bergman)**
- **'Smultronstället' (1957, dir. Ingmar Bergman)**
- **'The Shining' (1980, dir. Stanley Kubrick)**
- **'Computer Chess' (2013, dir. Andrew Bujalski)**