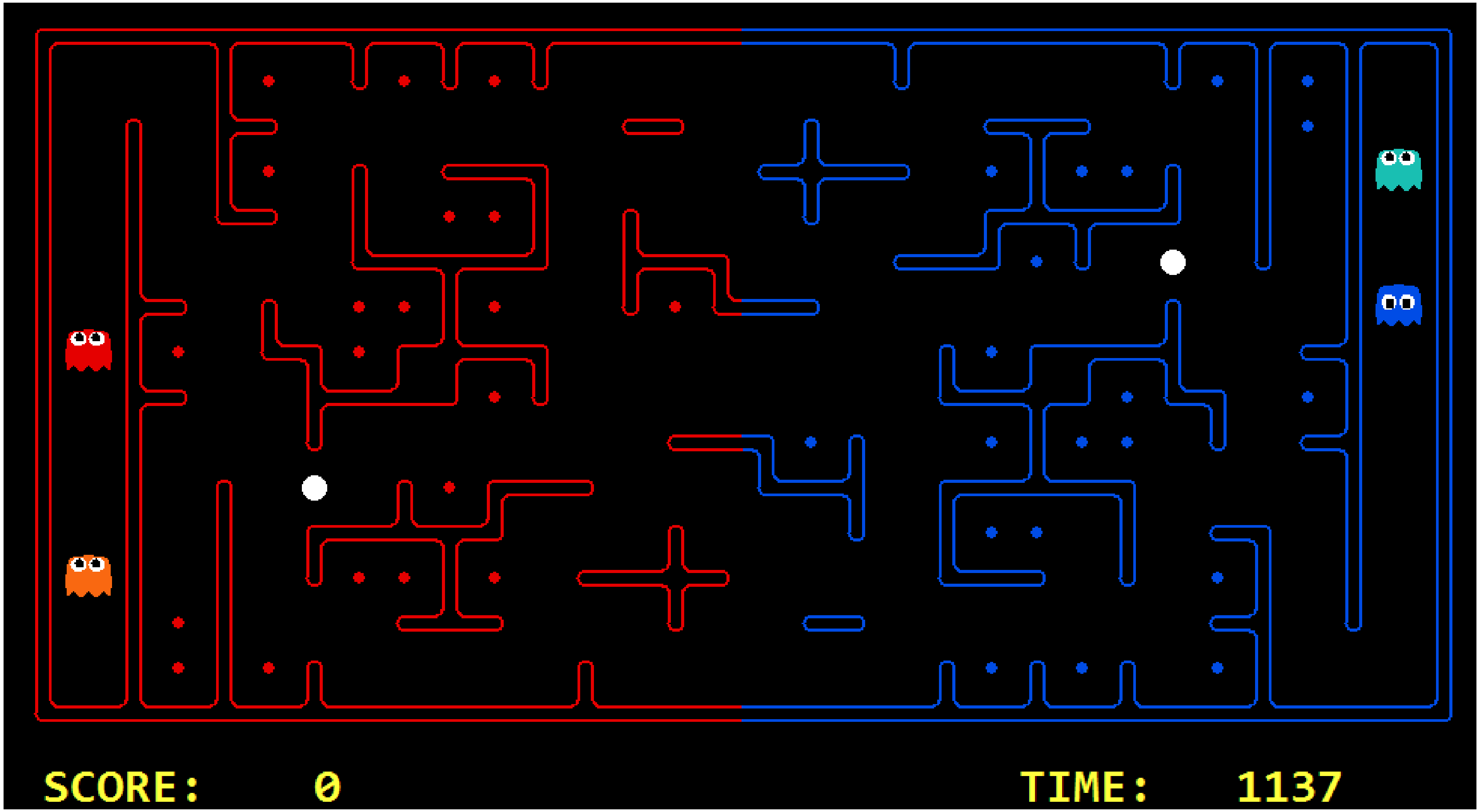
NOVA IMS Information Management School

Pac-Man project

Ivo Gonçalves
igoncalves@novaims.unl.pt

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Capture the food



Description

- Two teams with 2 agents each compete to capture the most food pellets
- Each team has a color: red or blue
- Each maze is divided into a red half and a blue half

Description

- The red half only has red food pellets and the blue half only has blue food pellets
- An agent in its own half is a ghost
- An agent in the opposing half is a Pac-Man

Description

- To score points an agent must:

    1. Go into the opposing half
    2. Capture food pellets
    3. Get back into its own half

Description

- When an agent carrying food pellets gets back into its own half, the captured food pellets are automatically deposited and the corresponding points are awarded
- One point is awarded for each food pellet deposited
- In the graphical user interface (GUI) the score shown is the difference between the red team points and the blue team points

Description

- This means that the score shown can have the following values and meanings:
  - 0, if both teams have deposited the same amount of food pellets
  - > 0, if the red team has deposited more food pellets than the blue team
  - < 0, if the blue team has deposited more food pellets than the red team

Description

- The game ends when:

   1. A team is able to capture and deposit at least N - 2 food pellets (where N is the total number of food pellets)

   2. The time ends

- If the time ends, the team with the most points (if any) wins

Description

- Each game consists of at most 1200 moves (300 per agent)

- The total number of moves left is labelled as “Time” in the GUI

Description

- A ghost can be in its normal state or it can be scared
- Any Pac-Man can be eaten by a normal state ghost
- If this happens the Pac-Man loses the food pellets that it was carrying (if any) and is sent back to its starting position
- These lost food pellets are distributed around the position where the Pac-Man was eaten

Description

- Power capsules exist on both halves of the maze

- The capsules on the red half can only be eaten by blue agents, while the capsules on the blue half can only be eaten by red agents

- When a Pac-Man eats a capsule every ghost from the opposing team becomes scared

Description

- Ghost remain scared for 40 moves if they are not eaten

- If a ghost is eaten it returns to its starting position as a normal state ghost

Code

- The code that we are going to use is based on “The Pac-Man Projects” from University of California, Berkeley

- http://ai.berkeley.edu/project_overview.html

- The code is in Python 2.7

Python 3.x vs. 2.7

- Python 3.0 broke backward compatibility
- print used to be a statement (in Python 3.x it is a function):

  Python 2.7: print ‘hello’

  Python 3.x: print(‘hello’)

Python 3.x vs. 2.7

- Division between integers used to be a floor division (in Python 3.x it is a floating point division):

Python 2.7: 5 / 2 returns 2

Python 3.x: 5 / 2 returns 2.5

Python 3.x: 5 // 2 returns 2

Python 3.x vs. 2.7

- range (among others) used to be a function that returned a list (in Python 3.x it is class that returns an iterable object):

Python 2.7: type(range(10)) returns <type ‘list’>

Python 3.x: type(range(10)) returns <class ‘range’>

Python 3.x vs. 2.7

- The change in range (as well as others) can be transparent depending on the code used

- The following codes produce the same result:

```
Python 2.7: for i in range(10):
                print i

Python 3.x: for i in range(10):
                print(i)
```

Building your agents

- The code for you agents will be added to the myTeam.py file

- By default, the code for your first agent goes in the AgentA class, and the code for your second agent goes in the AgentB class

Building your agents

- If needed, you can add some initialization code to the registerInitialState method of each agent class

- During the game loop the chooseAction method of each agent class is going to be called in order to receive the next action of each agent

Building your agents

- In this scenario, an action can be one of the following:
  - Move north
  - Move south
  - Move east
  - Move west
  - Stop

Building your agents

- If needed, you can add more classes or functions to the myTeam.py file

- Your final submission should be self-contained in the myTeam.py file

Maze

- The bottom left corner of a maze corresponds to the coordinate x = 0, y = 0

- The x coordinate grows from the left to the right

- The y coordinate grows from the bottom to the top

Useful information

- Who am I?

myIndex = self.index

Obtains the index of the self agent:

- 0 or 2 for red agents
- 1 or 3 for blue agents

Useful information

- Who is my teammate?

team = self.getTeam(gameState)

Returns a list with the indexes of the agents of my team:

- [0, 2] if the self agent is from the red team
- [1, 3] if the self agent is from the blue team

Useful information

- Who are my enemies?

enemies = self.getOpponents(gameState)

Returns a list with the indexes of the agents of the other team:

- [1, 3] if the self agent is from the red team
- [0, 2] if the self agent is from the blue team

Useful information

- Am I in the red or the blue team?

red = self.red

, or:

red = gameState.isOnRedTeam(self.index)

This boolean is:

- set to **True** if the self agent is in the red team
- set to **False** if the self agent is in the blue team

You can also identify the team by the index of the agent

Useful information

- Where am I?

myPosition = gameState.getAgentPosition(self.index)

Returns a tuple with the position (x, y) of the self agent

Useful information

- Where is my teammate?

```
teammatePosition = gameState.getAgentPosition(teammateIndex)
```

Returns a tuple with the position (x, y) of your teammate

You can always know where your teammate is

Useful information

- Where are my enemies?

```
enemy1Position = gameState.getAgentPosition(enemy1Index)
enemy2Position = gameState.getAgentPosition(enemy2Index)
```

Each call returns a tuple with the position (x, y) of a given enemy

You can only know the position of a given enemy if you or your teammate are within 5 squares (Manhattan distance) of that enemy

If that is not the case, **None** will be returned

Useful information

- What are my legal actions?

```
actions = gameState.getLegalActions(self.index)
```

Returns a list of strings with the legal actions for the self agent

'Stop' is always a valid action

Useful information

- Where are the food pellets to eat?

```
foodToEat = self.getFood(gameState).asList(True)
```

Returns a list of tuples with the positions (x, y) where food pellets are available to be eaten

This list can be empty, but never **None**

Useful information

- Where are the food pellets to protect?

```
foodToProtect = self.getFoodYouAreDefending(gameState).asList(True)
```

Returns a list of tuples with the positions (x, y) where food pellets are available to be eaten by the enemy

This list can be empty, but never **None**

Useful information

- Where are the power capsules to eat?

```
capsulesToEat = self.getCapsules(gameState)
```

Returns a list of tuples with the positions (x, y) where power capsules are available to be eaten

This list can be empty, but never **None**

Useful information

- Where are the power capsules to protect?

```
capsulesToProtect = self.getCapsulesYouAreDefending(gameState)
```

Returns a list of tuples with the positions (x, y) where power capsules are available to be eaten by the enemy

This list can be empty, but never **None**

Useful information

- Am I a ghost?

```
pacman = gameState.getAgentState(self.index).isPacman
```

This boolean is:

- set to **True** if the self agent is a Pac-Man
- set to **False** if the self agent is a ghost

Useful information

- Am I a scared ghost?

```
scared = gameState.getAgentState(self.index).scaredTimer
```

If the self agent is a scared ghost, this integer represents the number of moves left until the agent returns to its normal state

This number is zero if the ghost is not scared or if the self agent is a Pac-Man

Useful information

- What is the size of the maze?

```
width = self.getFood(gameState).width
height = self.getFood(gameState).height
```

Obtains the width and the height of the maze

Useful information

- What is the current score?

score = self.getScore(gameState)

Obtains the current score. This is not the same score as shown in the GUI. This score is the difference between the points of your team and the points of the other team.

You always want to maximize this value

Useful information

- Is that a wall?

```
wall = gameState.hasWall(x, y)
```

This boolean is:

- set to **True** if there is a wall in position x, y

- set to **False** if there is no wall in position x, y

Useful information

- How far is position (x1, y1) from position (x2, y2)?

d = self.getMazeDistance((x1, y1), (x2, y2))

Returns the distance from position (x1, y1) to position (x2, y2)

Useful information

- Am I carrying any food pellets?

```
food = gameState.getAgentState(self.index).numCarrying
```

Returns the number of food pellets that the self agent is carrying

Playoffs

- Each one of you will create a team of agents to compete in the playoffs
- The initial set of matches will be randomly selected
- Each match will consist of 3 games, each on a randomly generated maze

Playoffs

- The contestant that wins most games, wins the match
- In case of a tie, a digital coin toss will be performed

Testing different mazes

- You can test different mazes by passing the parameter “-l LAYOUT_FILE” to capture.py

- This loads the layout (or maze) from the file named LAYOUT_FILE in the layouts directory (LAYOUT_FILE does not need to include the .lay extension)

- To create a random maze use RANDOM (case-sensitive) as the LAYOUT_FILE parameter

Testing different mazes

- All available non-random layouts:

  alleyCapture
  bloxCapture
  crowdedCapture
  defaultCapture
  distantCapture
  fastCapture
  jumboCapture
  mediumCapture
  officeCapture
  strategicCapture
  tinyCapture

Running several games

- To run several games use the “-n N” parameter
- This runs N games and presents the win rate for each team at the end
- The default value is 1

Controlling agents with the keyboard

- You can control one or two agents with the keyboard by using the “--keys” parameter:

--keys0: controls agent 0 (first red agent)

--keys1: controls agent 1 (first blue agent)

--keys2: controls agent 2 (second red agent)

--keys3: controls agent 3 (second blue agent)

Controlling agents with the keyboard

- First keyboard player (arrow keys also work):

    w: north

    s: south

    d: east

    a: west

    q: stop

- Second keyboard player:

    i: north

    k: south

    l: east

    j: west

    u: stop