Imperial College London

DE2-COM2 (Computing 2)

Dyson School of Design Engineering

PAC-MAN group assignment

Imperial College London

Session Plan

- PAC-MAN group project details (90 min)
  - Project brief
  - Rules
  - DRAW week programming competition
- Priority Queue and Heap (30 min)
  - Exercises with Priority Queue with Binary Heap

Imperial College London

Rules

1. You cannot change the rules of the game PAC-MAN.
2. Each team must program one PacMan and one Ghost.
3. You need to submit your complete source code for inspection.
4. Your source code will be evaluated on quality* and documentation.
5. You need to present your code in front of everyone and explain how it works.
6. Any algorithms are allowed, provided that you can explain exactly how they work.
7. Each group participates with their code (PacMan + Ghost) in the programming competition in DRAW week.

Imperial College London

Outline

- Draw week goal
- Game description
  - Map + elements
  - Agents (Ghosts & PacMan)
- Environment
  - Visualisation
  - Layout, parsing
  - GamePlay
- Example

Imperial College London

Outline

- Draw week goal
- Game description
  - Map + elements
  - Agents (Ghosts & PacMan)
- Environment
  - Visualisation
  - Layout, parsing
  - GamePlay
- Example

Imperial College London

Draw week goal

- Group project
- Code the AI behavior for the Agents (PacMan + Ghost)
- Use obtained knowledge so far and the provided environment (visualisation, classes, rules etc)

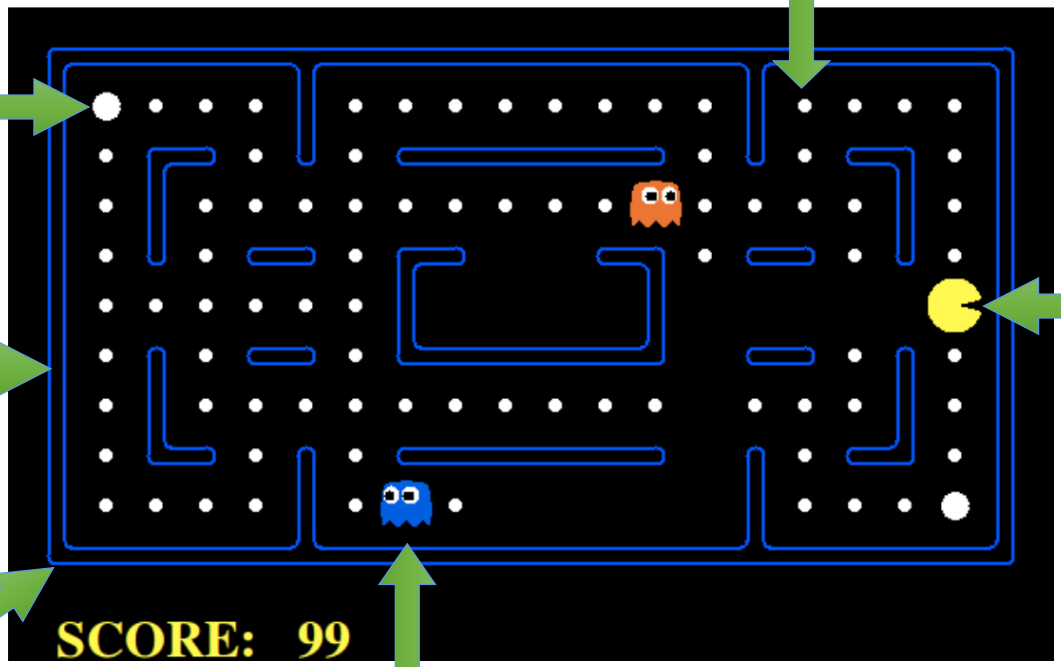
Imperial College London

Outline

- Draw week goal
- Game description
  - Map + elements
  - Agents (Ghosts & PacMan)
- Environment
  - Visualisation
  - Layout, parsing
  - GamePlay
- Example

Imperial College London

Game description



Food

Capsules

PacMan

Wall

Map origin (0,0)

Ghost

Imperial College London

Run the game

PyCharm:

python pacman.py

Imperial College London

Game description

PyCharm:

python pacman.py -h

Imperial College London

Game description

python pacman.py -h

```
Options:
  -h, --help          show this help message and exit
  -n GAMES, --numGames=GAMES
                      the number of GAMES to play [Default: 1]
  -l LAYOUT_FILE, --layout=LAYOUT_FILE
                      the LAYOUT_FILE from which to load the map layout
                      [Default: mediumClassic]
  -p TYPE, --pacman=TYPE
                      the agent TYPE in the pacmanAgents module to use
                      [Default: KeyboardAgent]
  -t, --textGraphics    Display output as text only
  -q, --quietTextGraphics
                      Generate minimal output and no graphics
  -g TYPE, --ghosts=TYPE
                      the ghost agent TYPE in the ghostAgents module to use
                      [Default: RandomGhost]
  -k NUMGHOSTS, --numghosts=NUMGHOSTS
                      The maximum number of ghosts to use [Default: 4]
  -z ZOOM, --zoom=ZOOM  Zoom the size of the graphics window [Default: 1.0]
  -f, --fixRandomSeed   Fixes the random seed to always play the same game
  -r, --recordActions   Writes game histories to a file (named by the time
                      they were played)
  --replay=GAMETOREPLAY
                      A recorded game file (pickle) to replay
  -a AGENTARGS, --agentArgs=AGENTARGS
                      Comma separated values sent to agent. e.g.
                      "opt1=val1,opt2,opt3=val3"
  -x NUMTRAINING, --numTraining=NUMTRAINING
                      How many episodes are training (suppresses output)
                      [Default: 0]
  --frameTime=FRAMETIME
                      Time to delay between frames; <0 means keyboard
                      [Default: 0.1]
  -c, --catchExceptions
                      Turns on exception handling and timeouts during games
  --timeout=TIMEOUT     Maximum length of time an agent can spend computing in
                      a single game [Default: 30]
```

Imperial College London

Game description

-l LAYOUT_FILE, --layout=LAYOUT_FILE
the LAYOUT_FILE from which to load the map layout
[Default: mediumClassic]

-p TYPE, --pacman=TYPE
the agent TYPE in the pacmanAgents module to use
[Default: KeyboardAgent]

-g TYPE, --ghosts=TYPE
the ghost agent TYPE in the ghostAgents module to use
[Default: RandomGhost]

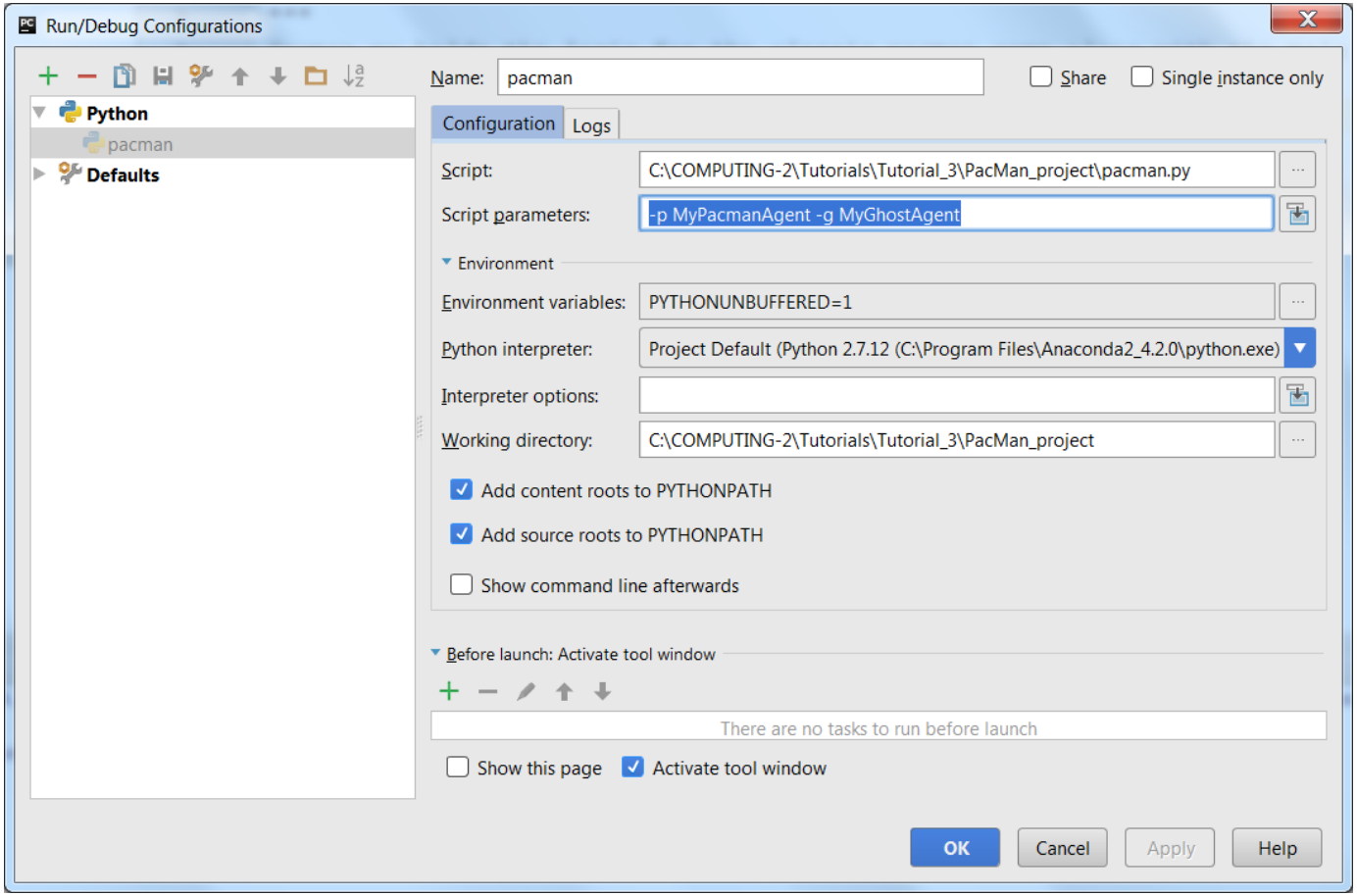
-k NUMGHOSTS, --numghosts=NUMGHOSTS
The maximum number of ghosts to use [Default: 4]

-z ZOOM, --zoom=ZOOM
Zoom the size of the graphics window [Default: 1.0]

Imperial College London

Command line options

Run → Edit Configuration:



Imperial College London

Outline

- Draw week goal
- Game description
  - Map + elements
  - Agents (Ghosts & PacMan)
- Environment
  - Visualisation
  - Layout, parsing
  - GamePlay
- Example

Imperial College London

Environment: Visualisation

The graphics are displayed using python's Tkinter library.

graphicsDisplay.py

- Defines environment colours
- Draws agents and elements (walls, food, capsules etc)
- Moves agents

graphicsUtils.py

- Defines polygons, squares, circles and other elements
- Handles keyboard input
- Updates graphics on-screen

Imperial College London

Environment: Utils

Additional support files:

utils.py

- contains some class definitions used.
- def chooseFromDistribution(distribution): chooses an action from a distribution.

textDisplay.py

- used for displaying states to user

Imperial College London

Environment: Layout parsing

layout.py

- converts layout .lay text file into level map
- main elements:

% - Wall
. - Food
o - Capsule
G - Ghost
P - PacMan

Environment: Layout parsing

examples are in folder “layouts”:
> mediumClassic.lay

```
%%%%%%%%%%%%%%%%%%%%
% o .  . . % . .  . . . . . .  % . . . . %
% . %%  . % .%%%%%% .  % . %% . %
% . %  . . . . . . . . . .  . . . % . %
% . %  . %% .%%     %% .  %%  . % . %
% . .  . . . . %G     G% .  .  . . . . %
% . %  . %% .%%%%%% .  %%  . % .%
% . %  . . . . . . . . . .  . . . % . %
% . %%  . % .%%%%%% .  % . %% . %
% . .  . . % . . . P  . . . .  % . . .  o %
%%%%%%%%%%%%%%%%%%%%
```

Imperial College London

GamePlay: PacMan

(option “-p”)

pacmanAgents.py

- *class LeftTurnAgent*
  - chooses action as to always turns left
- *class GreedyAgent*
  - chooses action that maximises the score
- *class MyAgent….*

keyboardAgents.py

- class KeyboardAgent & KeyboardAgent2
  - convert keyboard input to actions

Imperial College London

GamePlay: Ghost

(option “-g”)

ghostAgents.py

- *class GhostAgent:* inherits from Agent class (game.py).
- *class RandomGhost*
  - generate a uniformly distributed set of legal actions.
- *class DirectionalGhost*
  - implements Manhattan distance to evaluate the legal actions and generate the appropriate probability distribution over them.
  - when “scared” the ghost flees, otherwise goes towards PacMan.

➞ NOTE: Ghosts cannot stop or reverse (unless they reach a wall)

Imperial College London

GamePlay: controlling the game

game.py

- *class Agent*
- *class Directions:* defines the subsequent position based on current action.
- *class Configuration:* holds the position and heading direction of an agent.
- *class AgentState:* holds additional agent info (configuration, speed, scared, etc).
- *class Grid:* a 2-dimensional array of objects backed by a list of lists.
- *class Actions:* various action manipulators, conversion to vector, getting legal actions and neighbours etc.
- *class GameStateData:* initialises game info from data
- *class Game:* contains the main loop implemented by function “run”

Imperial College London

GamePlay: controlling the game

pacman.py

1. *Interface*
   - *class GameState:* keeps track of all the game elements (states, food, agents, configurations, score etc)
2. *Rules*
   - *class ClassicGameRules:* initialisation, end, game progress following
   - *class PacmanRules:* select&apply actions and eat food
   - *class GhostRules:* select&apply actions and eat&flee pacman
3. *GamePlay functions*
   - *def readCommand(argv):* gets the command line options
   - *def loadAgent(pacman, nographics):* loads the control strategy
   - *def runGames(layout, pacman, ghosts, display, numGames, record, numTraining = 0, catchExceptions=False, timeout=30):* initialises and runs the main game loop in *game.py*

Imperial College London

Outline

- Draw week goal
- Game description
  - Map + elements
  - Agents (Ghosts & PacMan)
- Environment
  - Visualisation
  - Layout, parsing
  - GamePlay
- Example

Imperial College London

Example test

Modify the script :
pacmanAgents.py

add a class:
class RandomAgent(Agent)

define a function:
def getAction(self, state)

which should make the PacMan perform random movements.

Imperial College London

Project task recap

- What you need to do:

Write the scripts
myPacmanAgents.py
and
myGhostAgents.py
to generate intelligent movements by modifying the *getAction* functions.

- Also, you can play with the layouts to test different environment configurations.

- As well as visualisation to change colours.

Imperial College London

Any Questions?

Dyson School of Design Engineering