Orbis Challenge Documentation

Release 1.0

Orbis Investment Management Limited

July 31, 2012

CONTENTS

1 Set-up Guide **3**
- 1.1 Get Started Guide . . . 3
- 1.2 Command Line Options . . . 5
- 1.3 Human Player Interface . . . 7
- 1.4 Submission and Grading . . . 7
- 1.5 Set-up Guide for Eclipse IDE . . . 7
- 1.6 Jython - Integrating the Java game and Python player . . . 14

2 Game Documentation **17**
- 2.1 Maze . . . 17
- 2.2 Ghosts . . . 20
- 2.3 Pacman . . . 22
- 2.4 Frame and Speed . . . 24
- 2.5 Levels and Scoring . . . 24

3 API Documentation **27**
- 3.1 Python API . . . 27
- 3.2 Java API . . . 34
- 3.3 Make Moves . . . 34

Python Module Index **37**

Index **39**

In this competition, you will be designing an AI player for the classic game Pacman!

You should first check out the Get Started Guide to configure your develop environment. Read the Game Documentation if you are not familiar with this game.

CHAPTER

ONE

SET-UP GUIDE

1.1 Get Started Guide

1.1.1 Choose Your Language

The AI player can be implemented in Python and Java. In regard to implementing the AI player for this game, the two languages are equivalent. So choose the one you are more comfortable with.

Download your development kit:

- Python AI DevKit
- Java AI DevKit

Unzip the file to your home folder and follow this guide to set up your develop environment.

1.1.2 Python Player Setup Guide

1. Install Java

Since the game itself is implemented in Java, you need **JRE 1.6 or higher** versions to run the game. Choose the steps based on your platform.

Mac OS X/Linux

Open up a terminal and run:

```
java -version
```

If the version number is 1.6.* or higher, you can continue to the next step. Otherwise, download and install the latest version of JRE from Oracle here. Choose "JRE Download" in "Java SE Downloads".

For Mac OS X users: you may already have the required JRE installed. In your Applications->Utilities, open "Java Preference" to select the requried Java version.

Windows

If you know how to check your JRE version on Windows, then you probably know what to do. If you don't know how to check your JRE version, or even what is JRE, just install JRE (version 1.7.0_05) using installation files in the folder "JRE" under "Python" directory. Choose the installation file according to your platform.

Make sure you install to "C:\Program Files\Java\jre7\".

2. Start the game with the sample player

Mac OS X/Linux

Go to the directory containing **pacman-python.jar**, run the following command in terminal to run the game with the sample AI player:

```
bash run.sh -ai
```

Windows

Go to the directory containing **pacman-python.jar**, run the following command in Command Prompt to run the game with the sample AI player:

```
run.bat -ai
```

If you encounter errors about Java path, please open the run.bat script and edit JAVA_HOME to the path you installed JRE to.

3. Implement your AI player!

Modify PacPlayer.py to implement your AI! **All Python code should stay in the directory "player".** To re-run the game, just use command in the previous step.

Before going too far, you are suggested to read the *Game Documentation* (page **??**) for details about how the game works.

Also, you definitely want to check out *Python API* (page 27) for the tools available.

Important: Do not change the file name or class name of PacPlayer.py.

1.1.3 Java Player Setup Guide

1. Install Java Development Kit (JDK)

You need **JDK 1.6 or higher** versions. Choose the steps based on your platform.

Mac OS X/Linux

Open up a terminal and run:

```
javac -version
```

If the version number is 1.6.* or higher, you can continue to the next step. Otherwise, download and install the latest version of JDK from Oracle here. Choose "JDK Download" in "Java SE Downloads".

Windows

If you know how to check your JDK version on Windows, then you probably know what to do. If you don't know how to check your JDK version, or even what is JDK, just install JDK (version 1.7.0_04) using installation files in the folder "JDK" under "Java" directory. Choose the installation file according to your platform.

Make sure you install JDK to "C:\Program Files\Java\jdk1.7.0_04\".

2. Start the game with the sample player

Mac OS X/Linux

Go to the directory containing the file **pacman-java.jar**, run the following command in terminal to compile and run with the sample AI player:

```
bash run.sh -ai
```

Windows

Go to the directory containing the file **pacman-java.jar**, run the following command in Command Prompt to compile and run sample AI player:

```
run.bat -ai
```

If you encounter errors about Java path, please check the run.bat and edit JAVA_HOME to the path you installed JDK to.

3. Implement your AI player!

Modify PacPlayer.java in the "player" directory to implement your AI. **All Java code should stay in the directory "player"**. The run script compiles PacPlayer.java and runs the game with the newly compiled AI. So, to re-run the game, simply use the same command in the previous step.

Before going too far, you are suggested to read the *Game Documentation* (page **??**) for details about how the game works.

Also, you definitely want to check out *Java API* (page 34) for the tools available.

Important: Do not change the file name or class name of PacPlayer.java

1.2 Command Line Options

When you start the game using run script in Terminal or Command Prompt, you can use options by adding them after the run script command:

on Mac OS X/Linux:

```
bash run.sh [options]
```

on Windows:

```
run.bat [options]
```

1.2.1 Use AI or Human Player

You can choose to play the game with your AI player or yourself using the keyboard! To use the AI player

on Mac OS X/Linux:

```
bash run.sh -ai
```

or on Windows:

```
run.bat -ai
```

To play the game yourself:

on Mac OS X/Linux:

```
bash run.sh -human
```

or on Windows:

```
run.bat -human
```

Press Space key to start or pause the game. Use arrow keys to move Pacman. See *Human Player Interface* (page 7) for detail.

1.2.2 Graphics Switch

Turning off GUI may help you test the AI more quickly. Just add:

```
-nonui
```

to the options.

Note: you won't be able to play the game as a human when the GUI is off.

1.2.3 Speed Control

The game speed (not to confuse with Ghost or Pacman's *Speed* (page 24)) can be changed through command line option.

To set the game speed, use the following in the command line options when starting the game:

```
-speed [an integer]
```

For example, use speed 20:

```
-speed 20
```

You can set speed value from 1 to 30. If you don't specify a speed value, a default value will be used.

1.2.4 Level Setting

To choose a specific level to start the game, use:

```
-level [an integer]
```

The lowerest the level is 1, which is the default starting level.

1.2.5 Cheating!

All games have cheating mode, and this one is no exception. You can cheat by turning on this option:

```
-cheat
```

This can help you experience different levels and mazes. However, we won't allow you to cheat when we grade your AI!

1.3 Human Player Interface

The human player interface cannot be used when the *Graphics Switch* (page 6) is off.

Use Space key to start/pause the game. You may also use the play/pause button on the right sidebar for this purpose.

Use Arrow keys to control Pacman.

You can change game speed real-time using "-" and "=" keys. Use "-" to decrease speed and "=" to increase speed.

1.4 Submission and Grading

1.4.1 How to submit your solution

1. Before submission, make sure all your code (source and class files) is inside the folder "player".
2. Rename the "player" folder to your team name.
3. Compress the folder, upload them here. Note we only accept .zip or .tar files. Make sure your submission is less then 5 MB.

1.4.2 How we grade your solution

We will grade your solution in -ai -nonui mode. We will run your solution 3 times and take the highest score. Please note that the total time allowed for each run is 5 minutes. If any of your run exceed 5 minutes, we will cut it off, and the scores you have received so far will be used.

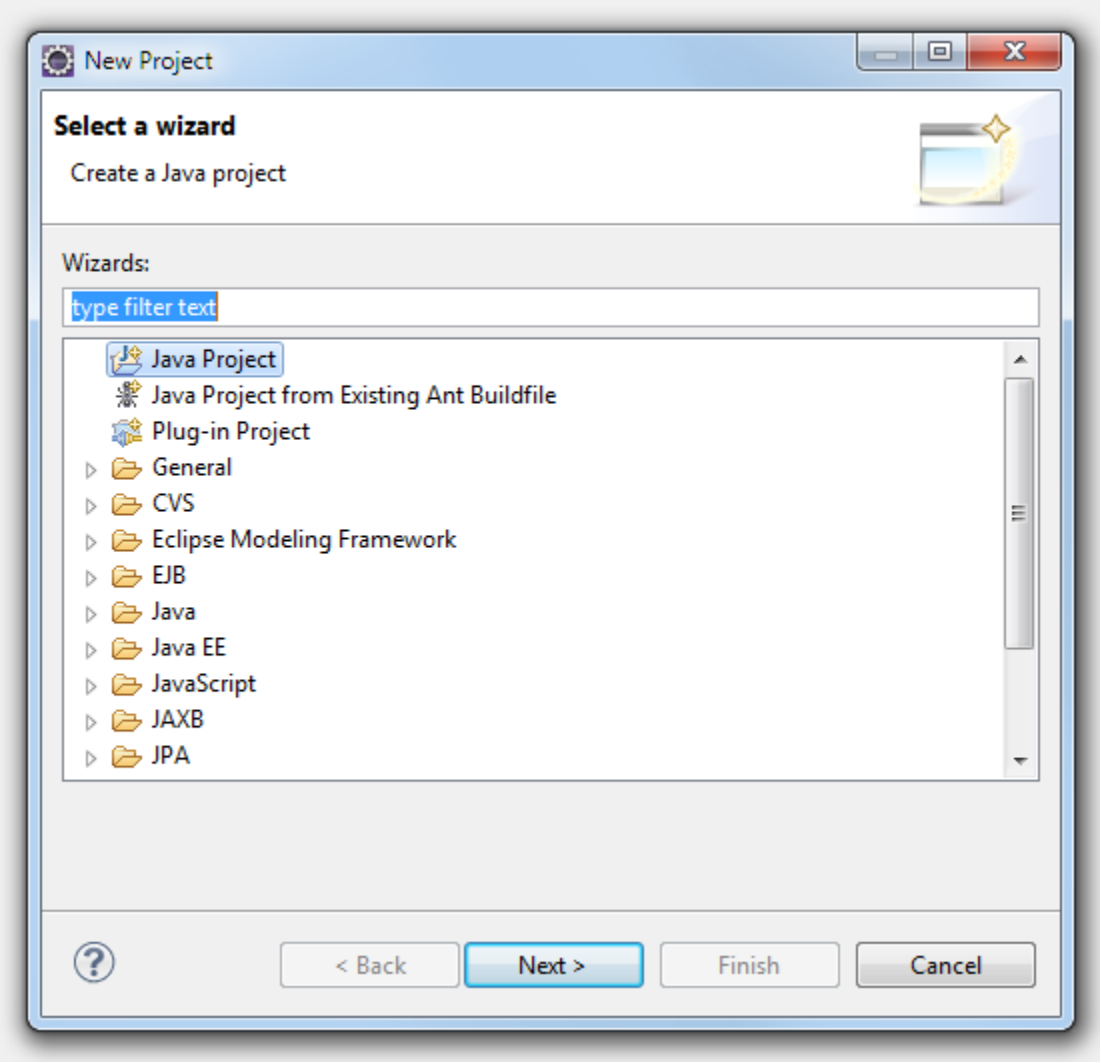
You can submit as many times as you wish, but only your last submission will be used.

1.5 Set-up Guide for Eclipse IDE

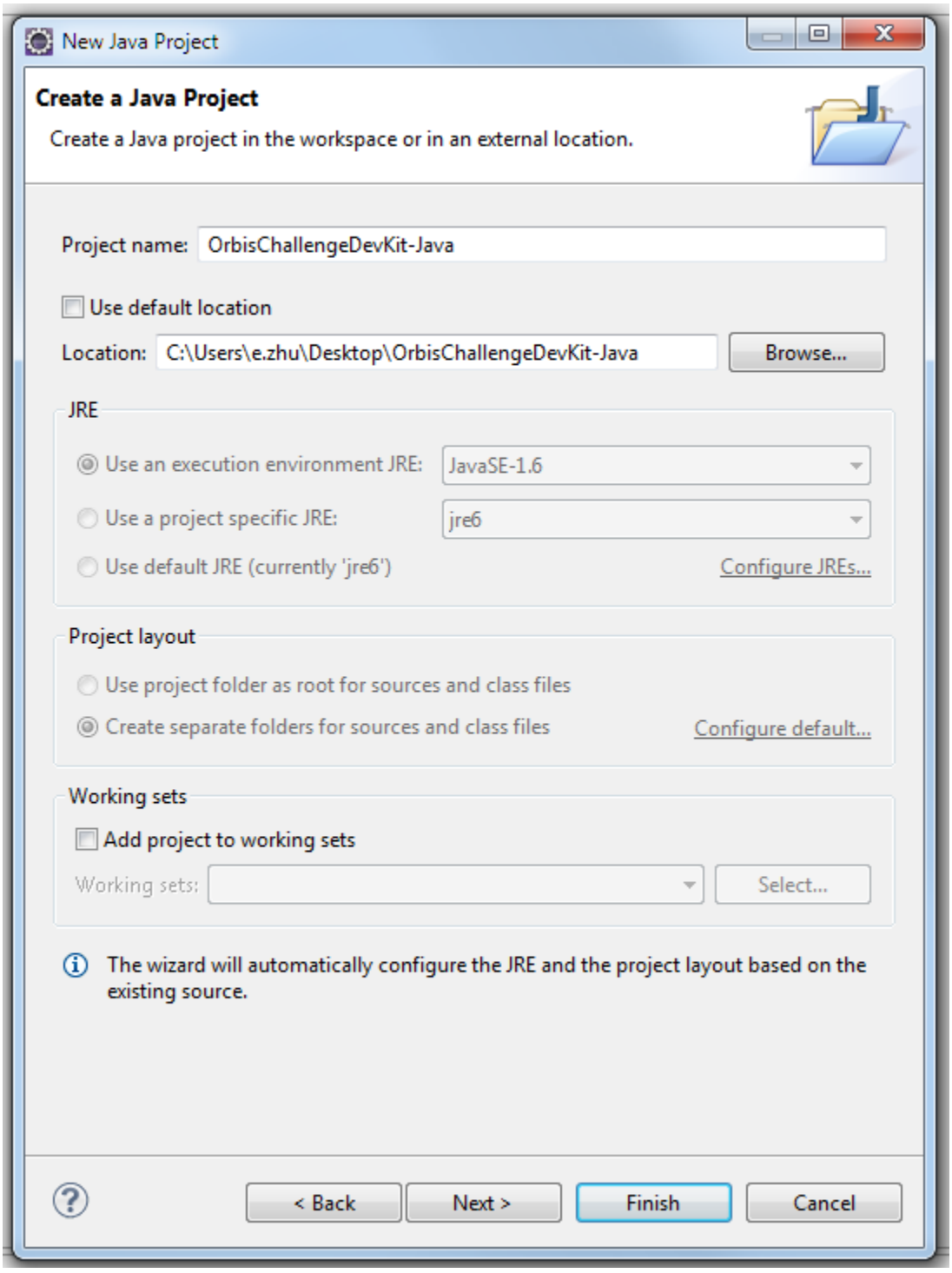
This guide for those of you who wish to use Eclipse IDE to develop your Java AI. Download Eclipse

Important: make sure your AI works when running from command line.

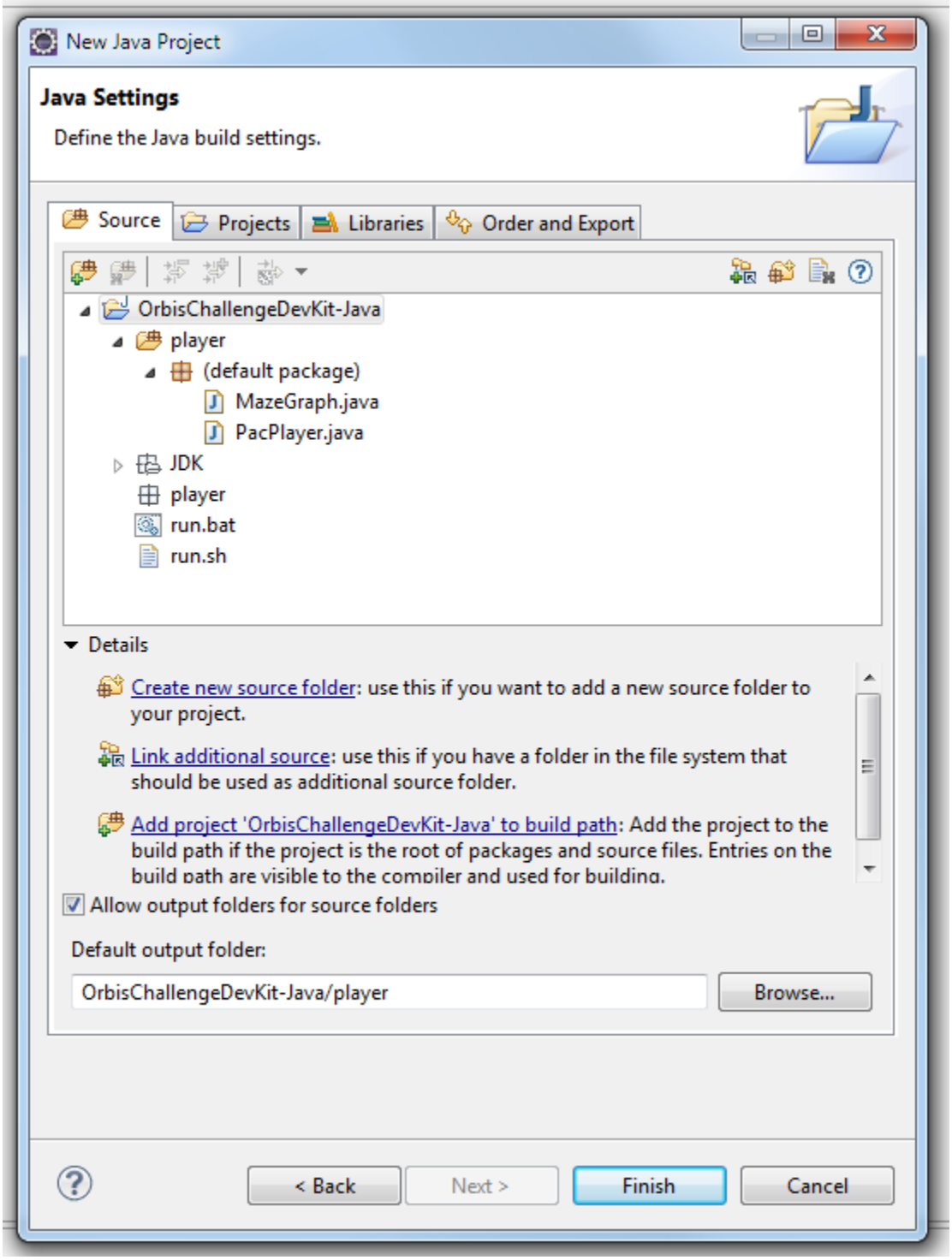
1. In the main menu, select Files->New->Project to create a new project. Use "Java Project".



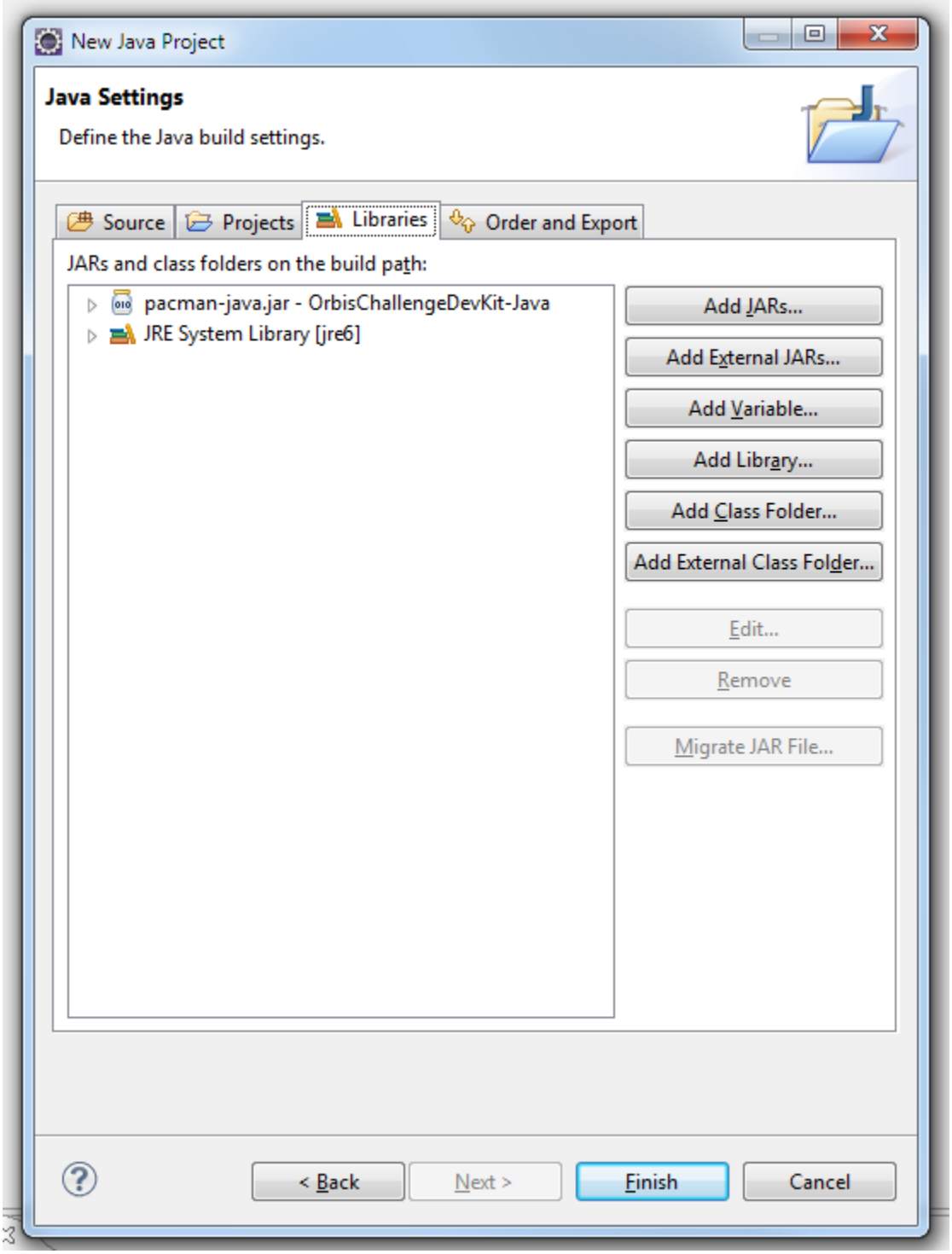
2. Uncheck "Use default location". Use the location of the devkit, as shown below. Make sure you are using "JavaSE-1.6" or "jre6" or higher versions (if not, you have to install JDK, refer to *Java Player Setup Guide* (page 4)). Leave other settings same as the image below.



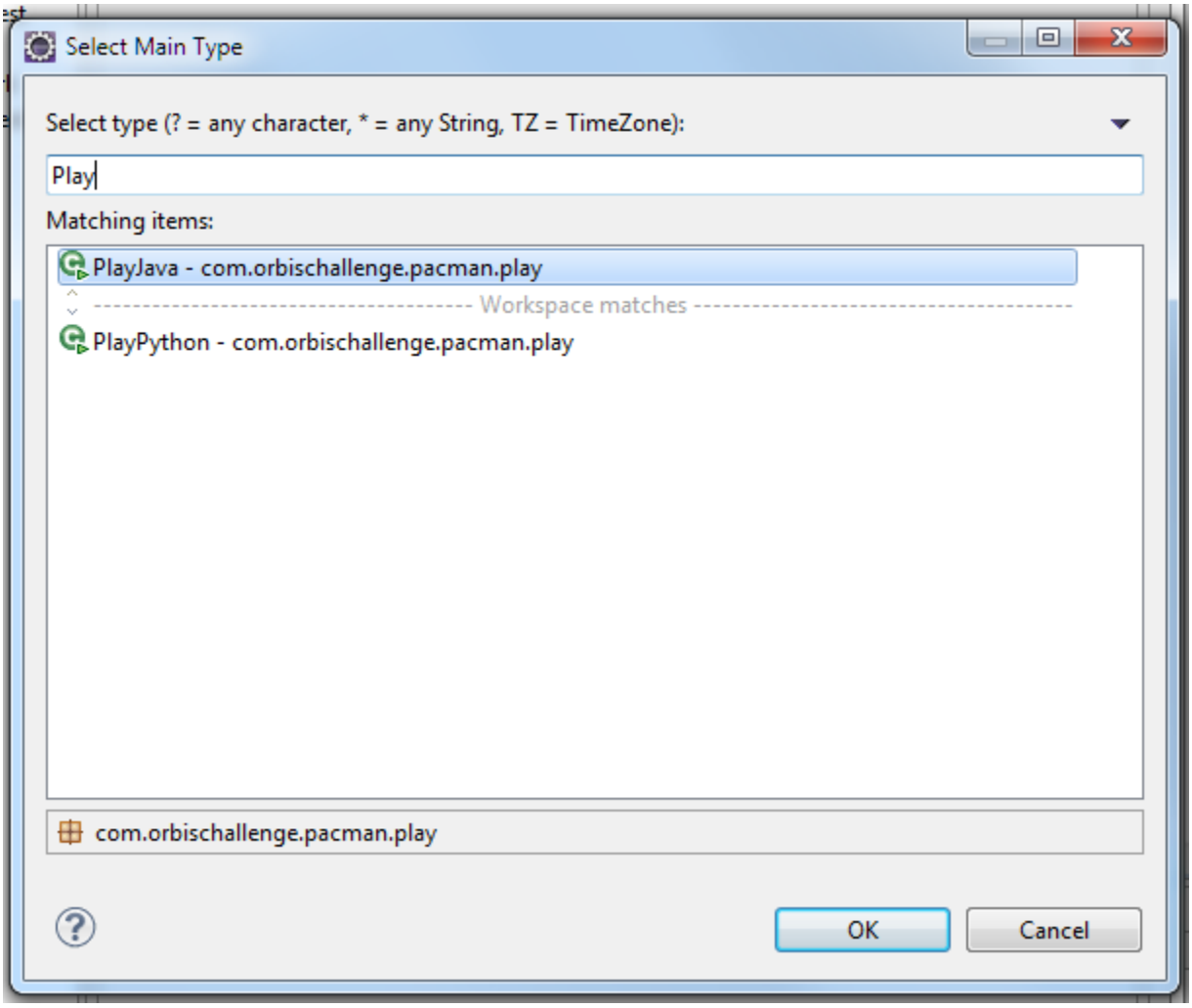
3. Make sure you have your setting looks like the below image: player folder should be both the source and output folder. Check the box “Allow output folders for source”. The default output folder should be set to the player folder. That is, all .class files should go into the player folder.



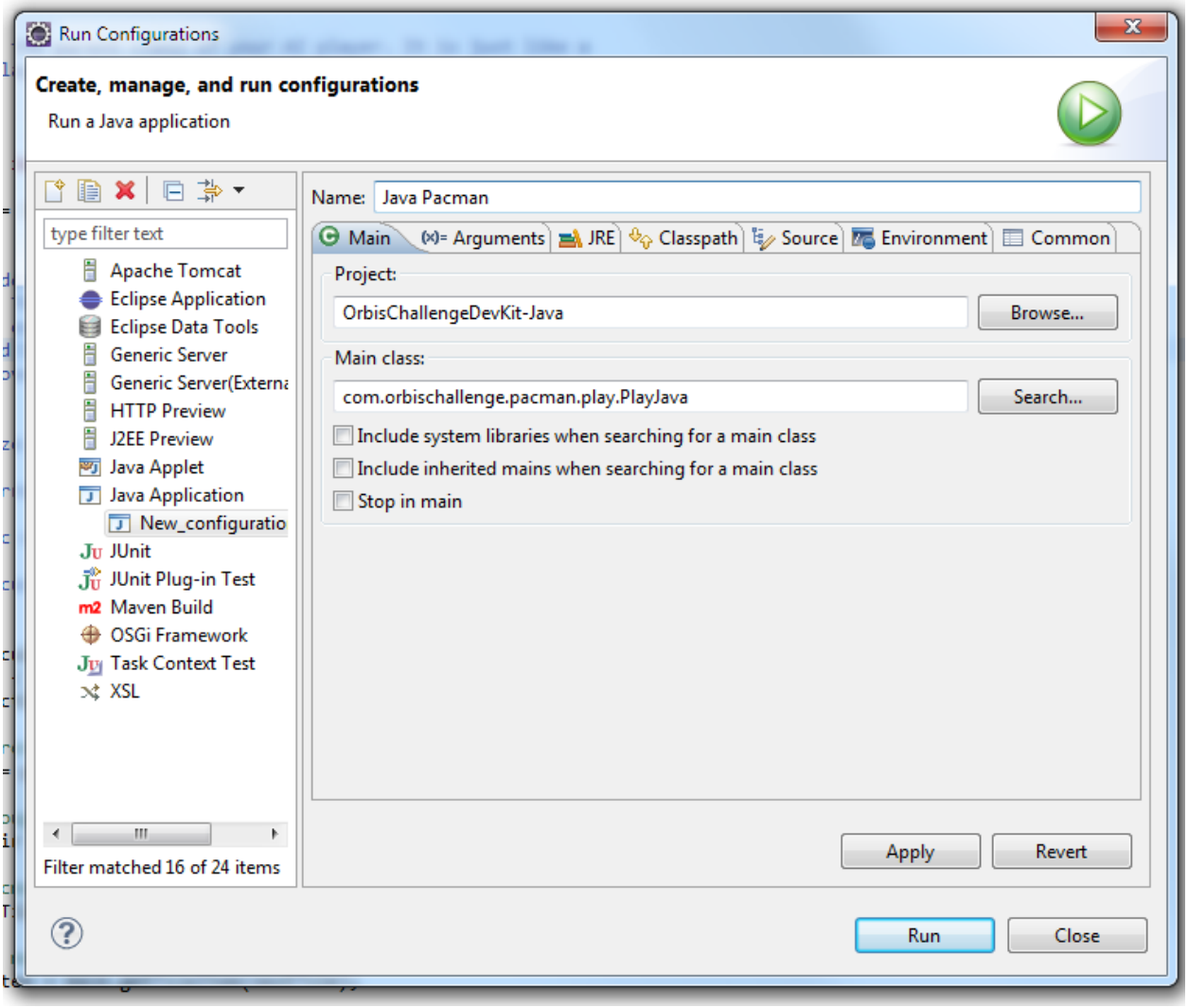
4. In Libraries tab, make sure you have pacman-java.jar added. If not, add it through “Add JARs...”. Click “Finish” to create this project.



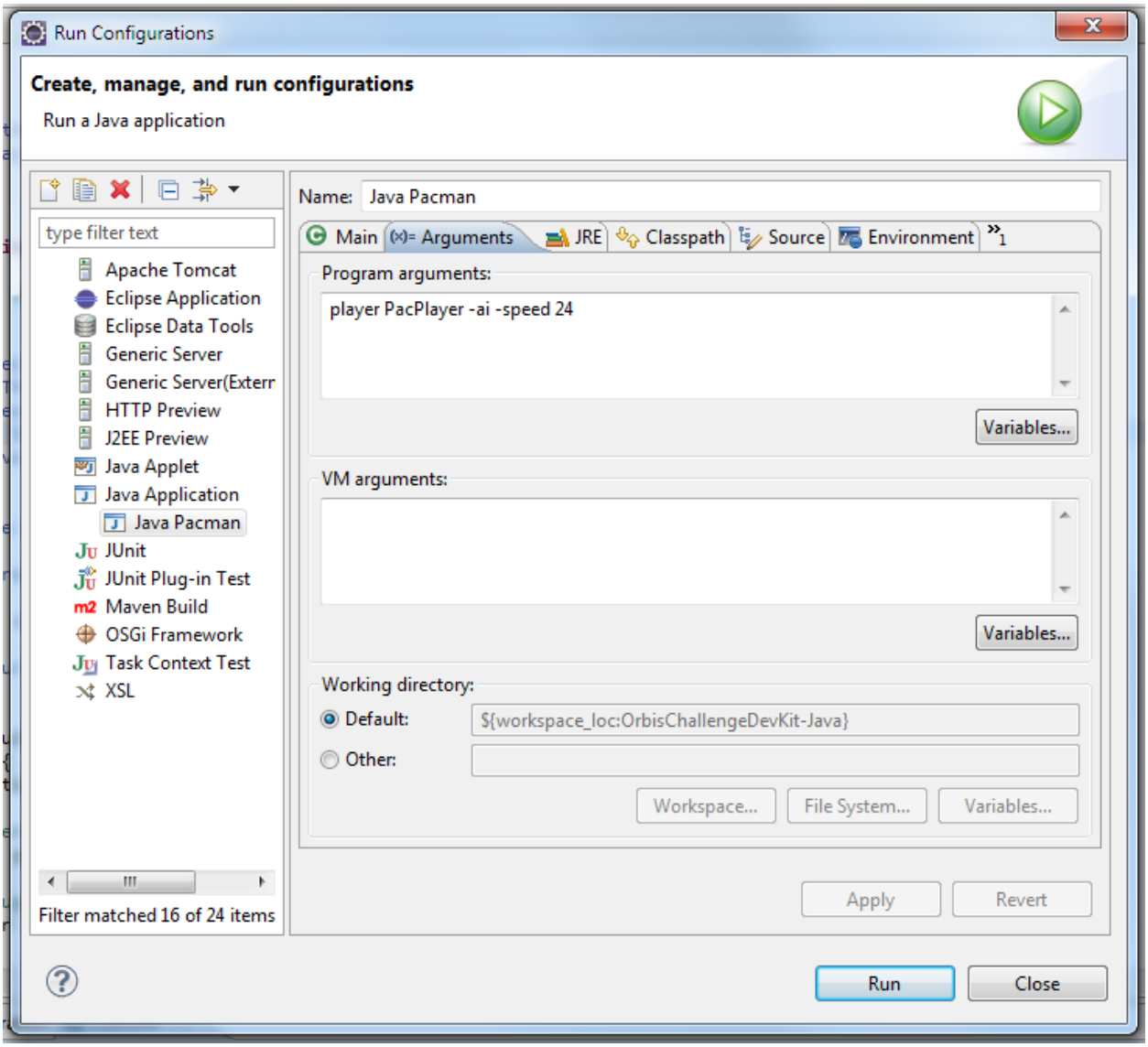
5. After you have successfully created the project. You may get some warning from Eclipse. Don't worry about them. Now you need to create a new run configuration. In the main menu, under Run, select Run Configuration. In the left-side bar, right-click "Java Application", select "New". Then, under "Main" tab, click "Search" to select the main class that is used to run the game. Use "PlayJava" as your main class. It should resides in the package "com.orbischallenge.pacman.play".



6. You can rename your run configuration to some meaningful name such as “Java Pacman”.



7. Under tab “Arguments”, type in “player PacPlayer -ai -speed 24” in “Program arguments”. The first argument is the directory containing the AI player. The second argument is the class name of the AI player. The rest are *Command Line Options* (page 5).



8. Lastly, make sure you have JRE6 (or higher) selected in Runtime JRE Tab. Click "Apply" to save this run configuration. Then, click "Run" to run the game with your AI player.

1.6 Jython - Integrating the Java game and Python player

The Python AI is really interpreted in Java Virtual Machine. This is made possible by the Jython Project. Jython is a different implementation of the Python language, other than the CPython you normally use.

Note that C modules are not available in Jython. Otherwise you probably don't need to worry about the differences in developing the AI.

1.6.1 Unit Test for Python AI

For this competition, Jython Library has been packaged into a .jar file in the Python DevKit. So you don't need to install Jython to run your AI. However, you should download and install it if you want to unit test your code.

When you do unit test, make sure you include pacman-python.jar in your sys.path. See the code snippet below.

```
# test.py
if __name__ == "__main__":
  # Only do this in unit test
  import sys
  import os
  # First we need to add our game into sys.path
  curr_dir = os.path.dirname(os.path.abspath(__file__))
  sys.path.append(os.path.join(os.path.dirname(curr_dir), 'pacman-python.jar'))

# Then we can start import the API modules, as they are in our sys.path
from com.orbischallenge.pacman.api.common import *
from com.orbischallenge.pacman.api.python import *

'''
Our AI module
'''

# Our unit test
if __name__ == "__main__":
  print MoveDir.values()
```

Now, you can use Jython to test your module directly from command line:

```
user $ /path/to/your/jython/executable test.py
```

To make your life easier, you probably want to add Jython to your user PATH.

CHAPTER

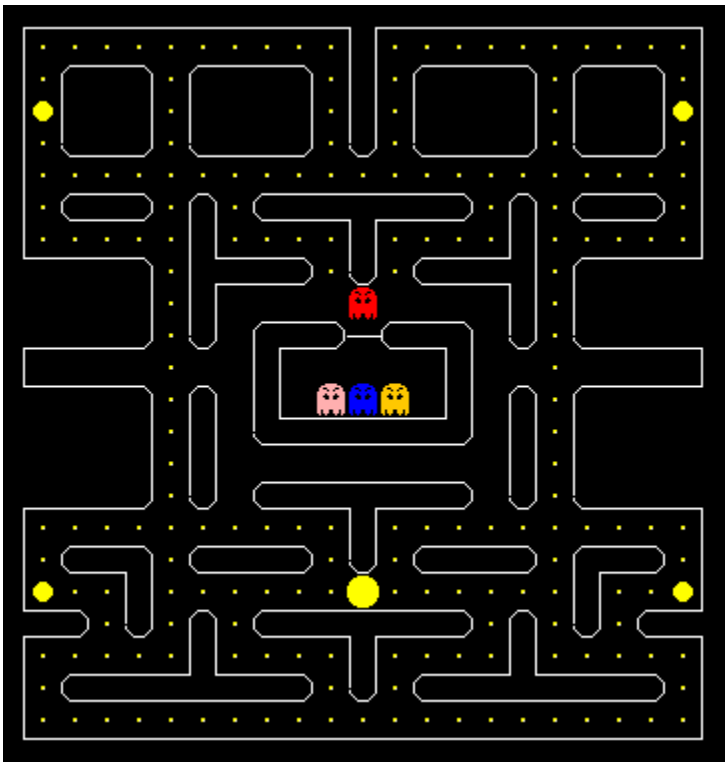
TWO

GAME DOCUMENTATION

2.1 Maze

The pacman game happens on the maze. There are different mazes for different levels.

This is the level one maze:



2.1.1 Maze Items

There are 6 different kinds of immobile items on the maze: *Blank* (page 17), *Wall* (page 18), *Dot* (page 18), *Power Dot* (page 18), *Door* (page 18), and *Teleport Tunnel* (page 18).

Blank

Blank is blank. The empty space. Sometime teleport tunnel can look like blank, be careful!

Wall

Walls are considered solid objects which neither pacman or ghosts can go through. A move in the direction of a wall is considered invalid and will not be executed by the game, and the original direction will be used instead. Also see Pacman's *Move* (page 22).

Dot

Dots are sweeties that give scores. The pacman's goal is to finish eating all the dots on the maze.

Power Dot

Power dots are energizers that help Pacman fight against the ghosts. The ghosts will turn into frightened state as pacman eats a power dot. However, the length of the effect can vary depending on the level and the number of times the ghosts have been frightened. They blink all the time to attract your attention!

Door

Door is a special kind of wall. Pacman cannot go through door, neither can ghosts in normal states. Only ghosts that are in the *Ghost House* (page 18) and ghosts hunted down by the pacman (in flee state) can go through door. Once a ghost leaves the ghost house through door, it cannot go back until it is hunted down.

Teleport Tunnel

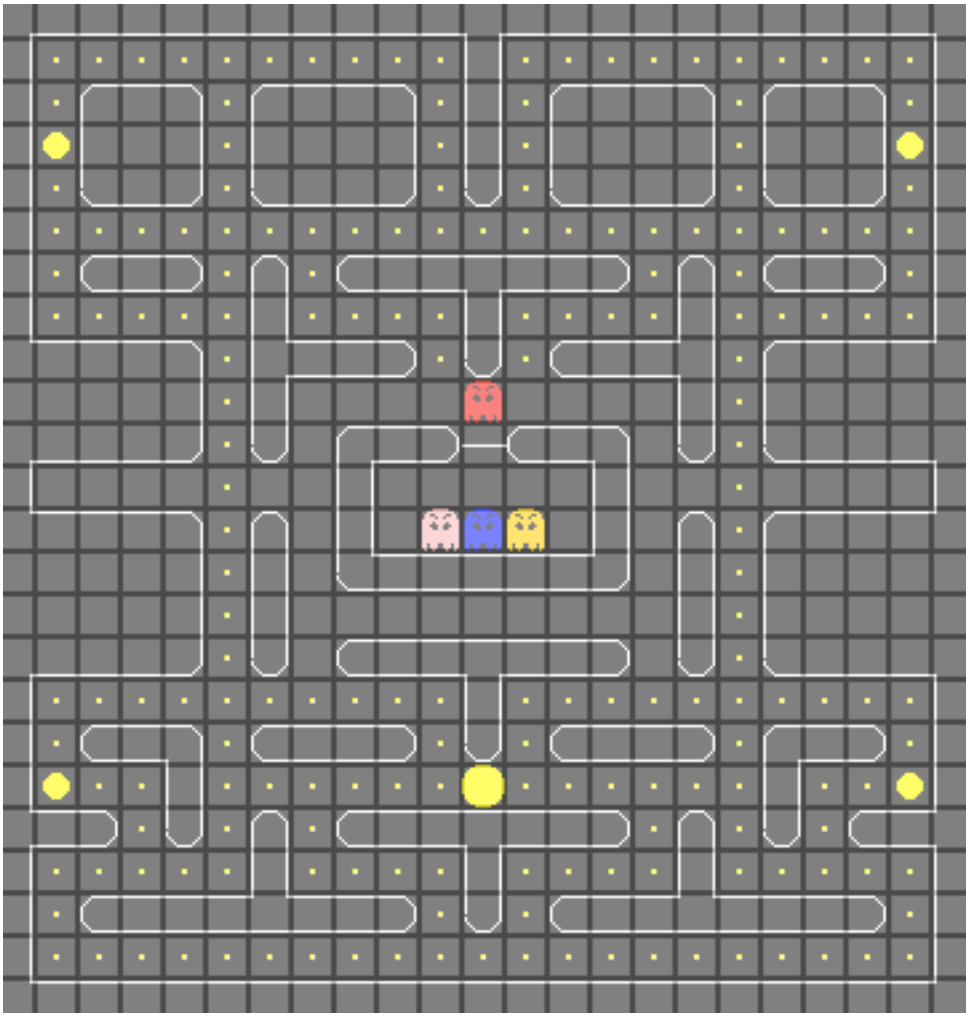
Teleport tunnels are not immediately noticeable - they look just like blank, but they have no dots on them. There are two of them on every maze. If Pacman or a ghost enters one tunnel, it will emerge from the other one. Ghosts can also use the tunnels.

2.1.2 Ghost House

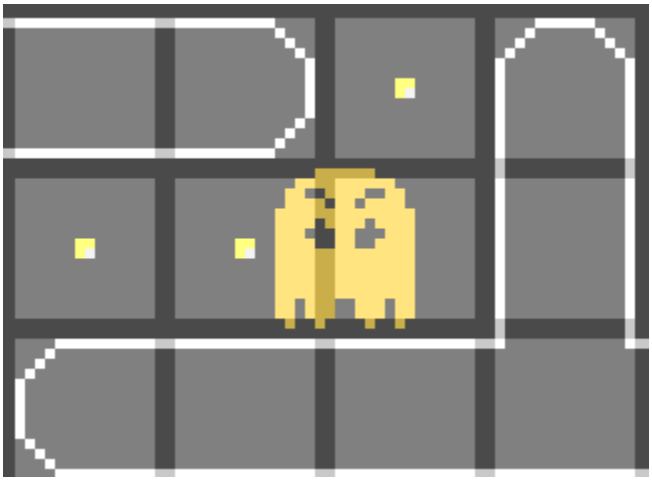
Ghost house is a special area on the maze surrounded by wall and its only entrance/exit is a door (See *Door* (page 18)). Pacman cannot go into the ghost house. Some ghosts stay in the ghost house at the start of each level (See *In house* (page 20)). When a ghost is hunted down, it will try to go back to the ghost house before remerging (See *Flee* (page 20)).

2.1.3 Tiles

The maze can be represented as a tiled grid. Each maze item occupies exactly one tile. The dimensions of a tile is equal to the dimensions of Pacman and a ghost. See the tiled maze below:



The dimensions of a tile is exactly 16 pixels in height and 16 pixels in width. The height and width of of the maze are multiples of 16.



When a ghost (or Pacman) is moving, it can span across more than one tile. However, its centre can only be in one tile at any moment (see *Pixel Coordinate* (page 20)).

2.1.4 Pixel Coordinate

The maze is really made up of pixels.

Every pixel on the maze has a 2D coordinate. The top-left pixel has coordinate (0, 0). The x coordinate increases from left to right, and the y coordinate increases from top to bottom.

The positions of ghosts and Pacman are determined using pixel coordinates. See ghosts' *Position* (page 21) and Pacman's *Position* (page 23).

2.2 Ghosts

Ghosts are immortals in the Pacman World. There are in total four of them, with different *Characters* (page 21). They always exist, however they can be in different *States* (page 20). Their overall objective is to disrupt Pacman's goal. A ghost may kill Pacman if they collide, or it may be hunted down, depending on its current state.

2.2.1 States

A ghost can be at only one state at any moment. Its state determines its current objective. Transitions between states may happen spontaneously.

In house

Whenever a ghost is inside the ghost house, it is at this state. As soon as a ghost finds its way outside of the ghost house (through *Door* (page 18)), it resumes pursuing Pacman.

Chaser

When a ghost is at this state, it is pursuing Pacman. However it may not target directly at him. See *Characters* (page 21) for different behaviours of ghosts. Pacman will die when he misfortunately collides with a chaser ghost.

Scatter

This is a mysterious state. Ghosts occasionally switch to this state during its pursuit. When a ghost is at this state, it wanders toward a corner of the maze and temporarily stops hunting Pacman. However there isn't much room for leisure, as it will come back very quickly. Also be aware that colliding with a scatter ghost is still lethal!

Frightened

Once Pacman eats a *Power Dot* (page 18), it obtains the power to hunt down ghosts, and all ghosts not in the ghost house will be frightened. They become powerless and change to shadows. Pacman can now hunt those ghosts. Collision with a frightened ghost turns it into *Flee* (page 20).

Flee

As a frightened ghost collides with Pacman, it's spirit shrinks into a smaller form. It is called the flee state. fleeing ghosts move very quickly but they have no power. Colliding with fleeing ghosts causes nothing to happen. fleeing ghosts stay in this state until they run back to the *Ghost House* (page 18), and become *In house* (page 20) ghosts again.

2.2.2 Characters

Blinky

Blinky is the red ghost, and he is the most aggressive one. He is the only ghost which stays outside of the ghost house when a level starts. At his chaser state, he will target directly at Pacman. Blinky can get very angry sometime. When angry, he charges at Pacman at full speed!

Pinky

Pinky is the pink ghost. She is not as aggressive as *Blinky* (page 21), but she can cut you loose unexpectedly. When she is at chaser state, her strategy seems to be heading Pacman off.

Inky

Inky is the blue ghost. He is the most mysterious one. There isn't much information available about him, but rumour says, when he works closely with *Blinky* (page 21), Pacman is severely at stake.

Clyde

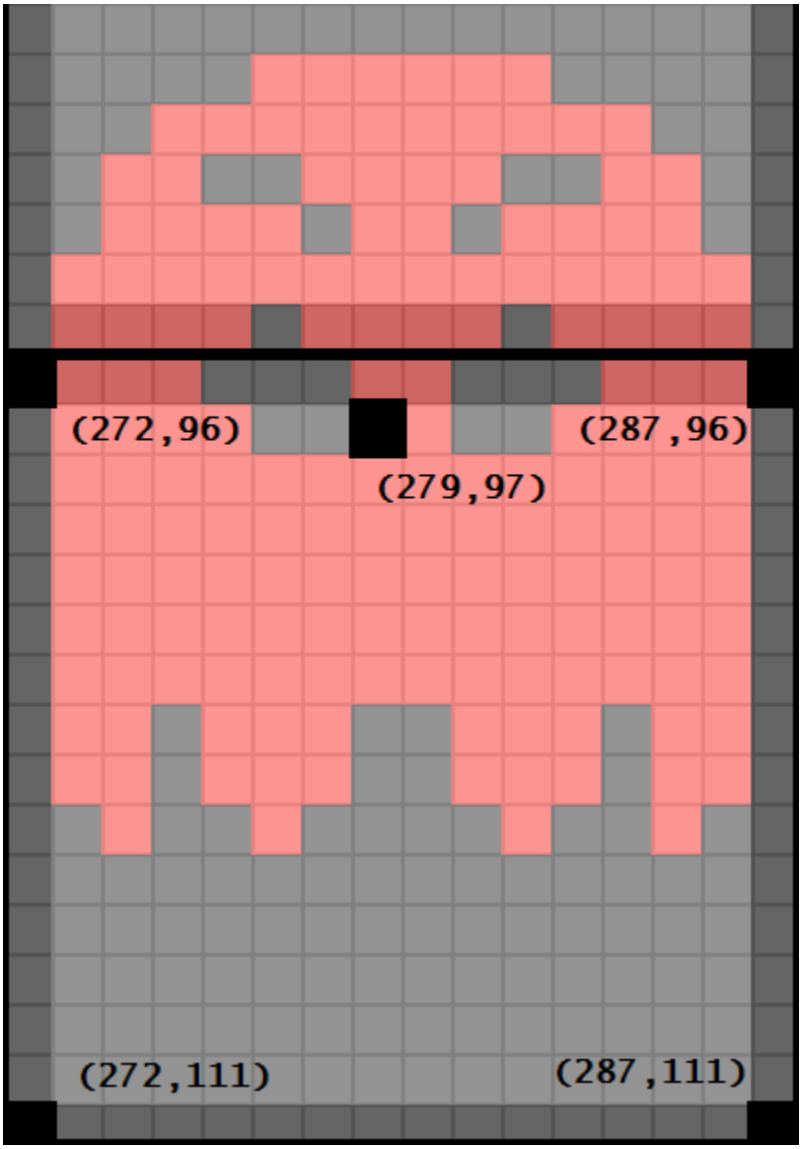
Clyde is the orange ghost. She doesn't seem to have too much interest in killing Pacman as long as he keeps a certain distance from her. When she is not pursing Pacman, she would wander around her own corner of the maze.

2.2.3 Position

Pixel Coordinate

The position of a ghost in *Pixel Coordinate* (page 20) is determined by its central pixel.

In the image of Blinky below, the central pixel's position is (279, 97), so the ghost's position is (279, 97).



Current Tile

A ghost may occupies multiple *Tiles* (page 18), but its current tile is always the one containing the central pixel. In the above image, its current tile has corner pixels (272, 96), (287, 96), (287, 111), and (272, 111).

2.3 Pacman

Pacman is the main character of the game. He is controlled by you, or your AI player. His goal is to finish eating all the *Dot* (page 18) on the maze, and survive!

2.3.1 Move

A move by Pacman is a displacement by one tile length, or 16 pixel, executed in 16 frame (see *Frame Concept* (page 24)). Thus, Pacman's speed is 1 pixel per frame.

Pacman can move in four directions: up, right, down, and left.

If Pacman is to change its direction in a move, and there is a wall or a door in the new direction, then the new direction will be considered invalid by the game, and the current direction will not be changed.

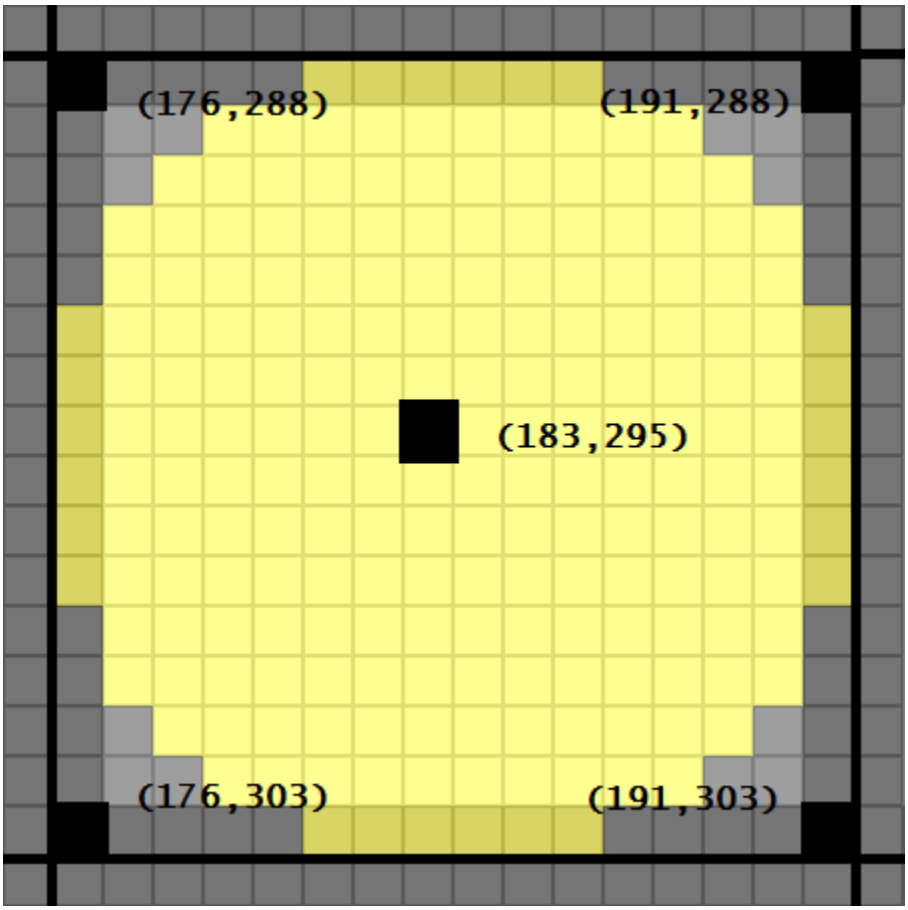
If Pacman is to keep its direction in a move, and there is a wall or a door in the current direction, then the move will be considered invalid by the game and will not be executed. This means Pacman will stop in the next frame.

2.3.2 Position

Pixel Coordinate

The position of Pacman in *Pixel Coordinate* (page 20) is determined by its central pixel.

For example, the level one initial position of Pacman is at (183, 295), represented by the marked central pixel.



Current Tile

Pacman's current tile is the one containing the central pixel. The corner pixels of Pacman's current tile in the above example are (176, 288), (191, 288), (191, 303), (176, 303). Similar to ghosts, Pacman can occupy multiple *Tiles* (page 18).

Since a Pacman's move is exactly one tile length, whenever you are asked to move Pacman, he is always at the center of a tile, occupying exactly one tile.

2.3.3 Collision with Ghosts

Let the position of Pacman be (u, v), and the position of a ghost be (x, y), both in *Pixel Coordinate* (page 20). If:

$$\|u - x\| <= 4$$
$$\|v - y\| <= 4$$

both equations are true, then there is a collision between Pacman and the ghost.

A collision with a ghost can kill Pacman if the ghost is at state *Chaser* (page 20), or *Scatter* (page 20). If the ghost is at *Frightened* (page 20) or *Flee* (page 20) state, the collision will not harm Pacman.

2.3.4 Life

Pacman will have **three** lives throughout the game. When he has lost all three lives, the game will be over. At the start of a new life during the game, Pacman and ghosts will be repositioned back to their initial locations on the current maze, and all other items on the maze will stay the same.

2.4 Frame and Speed

2.4.1 Frame Concept

Pacman and the ghosts do not know the concept of real time - time measured by your watch. Instead, it uses frames to measure the "time flow" in the game.

Your AI player is asked to make a move for Pacman before the execution of every 16 frames. See *Make Moves* (page 34).

Once the game get the move, it first lets ghosts and Pacman to move 1 frame. Then, it will evaluate the outcomes (e.g. collisions, eat a dot, etc.) of the moves in that frame. It will keep doing this for 16 frames.

After finishing executing the frames and if Pacman is still alive, the game asks for another move for Pacman and executes another 16 frames. The cycle repeats. As frames pass by, the game flows.

2.4.2 Speed

Speed is defined in pixels per frame. (See *Pixel Coordinate* (page 20))

Pacman moves at one pixels per frame throughout the game.

Ghosts' speed varies depending on their *States* (page 20), *Characters* (page 21) and *Levels* (page 24).

2.5 Levels and Scoring

2.5.1 Levels

Getting to higher levels is a good proof of your skill. Typically a new maze will be deployed as level increases, and difficulty will get progressively higher.

In order to appreciate your skill, there are infinite many levels. :)

Difficulty

An increase in difficulty is realized in terms of the followings:

- Increasing ghost overall cruising speed
- Shorter scattering period, and fewer occurrence of *Scatter* (page 20) states
- Shorter *Frightened* (page 20) period
- More aggressive ghost *Characters* (page 21)

2.5.2 Scoring

The score is accumulative throughout the game. Initial score is 0. Scores are added at the end of a frame based on the outcome of that frame (See *Frame Concept* (page 24)). Score increment is a function of level. Expect higher score reward in higher level. See outcomes versus score increment below.

Eat a dot

Dot (page 18) aren't just for scores, you need to finish all the dots on the maze to complete a level.

$$score\ increment = 1 + (level - 1) * 2$$

Eating a power dot

Eating a *Power Dot* (page 18) gives you twice many scores as eating a dot.

$$score\ increment = 2 + (level - 1) * 4$$

Hunting a ghost

When Pacman hunts down a ghost - colliding with a *Frightened* (page 20) ghost, scores will be awarded.

$$score\ increment = 50 + (level - 1) * 20$$

Completing a level

You will earn scores for completing a level. However you will be penalized based on the total number of frames you used to complete the level (see *Frame Concept* (page 24)).

$$score\ increment = max\{[500 + (level - 1) * 500 - total\ frames\ used/1000], 0\}$$

CHAPTER

THREE

API DOCUMENTATION

API stands for Application Programming Interface. This the only interface your AI and the game can communicate. It also provides you a set of methods and constants, which you cannot and/or probably don't want to write.

Yes, many of those are give-aways from us, in hope to save your time!

3.1 Python API

This API contains constants and methods of classes which can be used by the Python AI player. Your AI player class should extends the `Player` (page 33) class.

All Python API modules can be imported using Python import:

```
from com.orbischallenge.pacman.api.common import *
from com.orbischallenge.pacman.api.python import *
```

3.1.1 MoveDir

These are the directions used by Ghosts and Pacman to make their moves.

`MoveDir.`**`UP`**
: Representing an upward direction

`MoveDir.`**`RIGHT`**
: Representing a rightward direction

`MoveDir.`**`DOWN`**
: Representing a downward direction

`MoveDir.`**`LEFT`**
: Representing a leftward direction

`MoveDir.`**`values`**`()`
: Returns an array object containing the `MoveDir` (page 27) objects (one for each) in above order.

To convert the array into a Python list, you can use:

```
MoveDir_List = list(MoveDir.values())
```

To represent vectors and positions on the maze, we also use `2-Tuple` - tuples of 2 integers. The first integer is the horizontal component, the second integer is the vertical component. For example, directions on the maze can be represented as:

```
(0, -1) # upward unit direction vector
(0, 1) # downward unit direction vector
(1, 0) # rightward unit direction vector
(-1, 0) # leftward unit direction vector

(32, 0) # a direction vector

(64, 32) # ! this is not a direction vector, since we cannot move in that direction.
```

3.1.2 MazeItem

See *Maze Items* (page 17). Each `MazeItem` (page 28) object occupies exactly one tile of the maze.

`MazeItem.`**`BLANK`**
Representing a *Blank* (page 17) tile on the maze

`MazeItem.`**`WALL`**
Representing a *Wall* (page 18) tile

`MazeItem.`**`DOOR`**
Representing a *Door* (page 18) tile

`MazeItem.`**`DOT`**
Representing a *Dot* (page 18) tile. After Pacman eatting the dot, this tile will turn into a blank tile.

`MazeItem.`**`POWER_DOT`**
Representing a *Power Dot* (page 18) tile. After Pacman eatting the power dot, this tile will turn into a blank tile.

`MazeItem.`**`TELEPORT`**
Representing a *Teleport Tunnel* (page 18) tile.

`MazeItem.`**`values`**`()`
Returns an array object containing the `MazeItem` (page 28) objects (one for each) in above order.

To convert the array into a Python list, you can use:

```
MazeItem_List = list(MazeItem.values())
```

3.1.3 Maze

We also use `2-Tuple` to represent a position on the maze. For example:

```
(184, 123) # a pixel coordinate

(3, 9) # a pair of tile indexes
```

Notice that a pixel position and a tile position are very different, although they both use the same tuple representation.

You can use the following methods of a `Maze` (page 28) object

`Maze.`**`get_pixel_item`**`(`*`pixel`*`)`
Get the `MazeItem` (page 28) object at the tile containing the pixel, a Tuple (x, y). See *Pixel Coordinate* (page 20) for the meaning of (x, y).

Parameters **pixel** – `2-Tuple`, (x, y) in pixel coordinate

Return type `MazeItem` (page 28)

Maze.**get_tile_item**(*tile*)
Get the MazeItem object at a give tile indexes (column index, row index), see *Tiles* (page 18). This method is essentially a getter for the 2D list returned by `to_2d_dist()`.

Parameters **tile** – `2-Tuple`, (column index, row index)

Return type `MazeItem` (page 28)

Maze.**to_2d_list**()
Get a matrix (2D List) representation of the tiled maze. Each tile will be an element of the matrix.

Return type A `List` of `List` (rows) of `MazeItem` (page 28) objects.

Maze.**width**()
Get the width of the maze in pixel

Return type `Integer`

Maze.**height**()
Get the height of the maze in pixel

Return type `Integer`

Maze.**dots_count**()
Get the number of uneaten dots on the maze

Return type `Integer`

Maze.**get_power_dots_tiles**()
Get a list of tiles which contain available power dots.

Return type `List`

Maze.**accessible_neighbours**(*tile*)
Get accessible (to Pacman) neigbouring tiles of a given tile. Only directly adjacent tiles are neighbouring tiles.

Parameters **tile** – `2-Tuple`, (column index, row index)

Return type `List`

Maze.**is_accessible**(*tile*)
Check if a given tile is accessible to Pacman

Parameters **tile** – `2-Tuple`, (column index, row index)

Return type `Boolean`

Maze.**is_intersection**(*tile*)
Check if a given tile is an intersection. An intersection must be accessible to Pacman and has 3 or 4 directions going out of it. The directions must also be accessible to Pacman.

Parameters **tile** – `2-Tuple`, (column index, row index)

Return type `Boolean`

Maze.**is_corner**(*tile*)
Check if the given tile is a corner paassage. A corner must be accessible to Pacman and has only 2 directions going out of it. The 2 directions must be perpendiculars and accessible to Pacman.

Parameters **tile** – `2-Tuple`, (column index, row index)

Return type `Boolean`

Maze.**is_straight**(*tile*)
Check if the given tile is a straight passage. A straight passage tile must be accessible to Pacman and has only 2 directions going out of it. The 2 directions must be parallels (opposite directions) and accessible to Pacman.

Parameters **tile** – `2-Tuple`, (column index, row index)

Return type `Boolean`

`Maze.`**`is_dead_end`**(*tile*)
Check if the given tile is a dead end. A dead end tile must be accessible to Pacman and has only 1 direction going out of it. The direction must be accessible to Pacman.

Parameters **tile** – `2-Tuple`, (column index, row index)

Return type `Boolean`

3.1.4 GhostState

This module contains the states objects that are used by ghosts.

`GhostState.`**`IN_HOUSE`**
Representing the state at which ghosts are in the ghost house. See *In house* (page 20).

`GhostState.`**`CHASER`**
Representing the state at which ghosts are pursuing Pacman. See *Chaser* (page 20).

`GhostState.`**`SCATTER`**
Representing the state at which ghosts are wandering toward the corners of the maze. See *Scatter* (page 20).

`GhostState.`**`FRIGHTEN`**
Representing the *Frightened* (page 20) state of a ghost when Pacman eats a *Power Dot* (page 18).

`GhostState.`**`FLEE`**
Representing the state at which a ghost running fast toward the ghost house. See *Flee* (page 20).

`GhostState.`**`values`**()
Returns an array object containing the `GhostState` (page 30) objects (one for each) in above order.

3.1.5 GhostName

This module contains all the names of ghosts.

`GhostName.`**`Blinky`**

`GhostName.`**`Pinky`**

`GhostName.`**`Inky`**

`GhostName.`**`Clyde`**

`GhostName.`**`values`**()
Returns an array object containing the `GhostName` (page 30) objects (one for each) in above order.

An example of using `GhostName` (page 30):

```
if ghost.getName() == GhostName.Blinky:
  # Do something
```

3.1.6 Ghost

You can use the following methods of a `Ghost` (page 30) object

`Ghost.`**`center`**()
Get the ghost's position in *Pixel Coordinate* (page 20). (x, y)

Return type 2-Tuple

Ghost.**tile**()
Get the ghost's position in *Tiles* (page 18) indexes. (column index, row index)

Return type 2-Tuple

Ghost.**state**()
Get the current state of the ghost.

Return type GhostState (page 30)

Ghost.**name**()
Get the name of the ghost.

Return type GhostName (page 30)

Ghost.**dir**()
Get the current direction of the ghost.

Return type MoveDir (page 27)

Ghost.**frames_till_recover**()
Returns the number of frames till the ghost recovering from *Frightened* (page 20) state. If the current state of the ghost is not frightened, the method will return 0.

Return type Integer

Ghost.**distance_to_pac**(*pac*)
Calculate the euclidian distance (the norm) from the center of this ghost to the center of Pacman, in pixels.

Return type Float

3.1.7 Pac

You can use the following methods of Pacman.

Pac.**center**()
Get Pacman's position in *Pixel Coordinate* (page 20). (x, y)

Return type 2-Tuple

Example:

```
print pac.center()

>> (39, 23)
```

Pac.**tile**()
Get Pacman's position in *Tiles* (page 18) indexes. (column index, row index)

Return type 2-Tuple

Example:

```
print pac.tile()

>> (2, 1)
```

Pac.**dir**()
Get the current direction of Pacman.

Return type MoveDir (page 27)

Pac.**possible_dirs**()
Get all possible directions Pacman can currently move, i.e. there is no *Wall* (page 18) or *Door* (page 18) on the neighbouring tiles in those directions. This method does not consider the positions of Ghosts.

Return type A list of MoveDir (page 27)

3.1.8 PyUtil

This module contains some useful functions which may help you navigate the maze.

PyUtil.**get_vector**(*Dir*)
Get a unit vector representation of a given MoveDir object. One of (1, 0), (-1, 0), (0, 1), and (0, -1).

Parameters **Dir** – a MoveDir object

Return type 2-Tuple

PyUtil.**get_MoveDir**(*vector*)
Get a MoveDir object corresponding to a given direction vector.

Parameters **vector** – 2-Tuple, must be a direction vector

Return type MoveDir

Example:

```
vector = (0, -16)
print PyUtil.get_MoveDir(vector)

>> UP
```

PyUtil.**vector_add**(*u*, *v*)
Vector addition.

Parameters

- **u** – 2-Tuple
- **v** – 2-Tuple

Return type 2-Tuple

Example:

```
u = (41, 71)
v = (-16, 0)
print PyUtil.vector_add(u, v)

>> (25, 71)
```

PyUtil.**vector_sub**(*u*, *v*)
Vector subtraction.

Parameters

- **u** – 2-Tuple
- **v** – 2-Tuple

Return type 2-Tuple

PyUtil.**dot_product**(*u*, *v*)
Caculate dot product of two vectors.

Parameters

- **u** – 2-Tuple
- **v** – 2-Tuple

Return type Integer

PyUtil.**get_perpendiculars**(*vector*)

Get perpendicular unit direction vectors of a given direction vector

Parameters **vector** – 2-Tuple, must be a direction vector

Return type a List of 2-Tuple

Example:

```
vector = (30, 0)
print PyUtil.get_perpendiculars(vector)

>> [(0, 1), (0, -1)]
```

Player.**euclidean_distance**(*u*, *v*)

Caculate the euclidean distance between two points. This method should work for tiles coordinate and pixel coordinate, but you have to keep the inputs consistent.

Parameters

- **u** – 2-Tuple
- **v** – 2-Tuple

Return type Float

Player.**manhattan_distance**(*u*, *v*)

Caculate the manhattan distance between two points. Check out here for detail. This method should work for tiles coordinate and pixel coordinate, but you have to keep the inputs consistent.

Parameters

- **u** – 2-Tuple
- **v** – 2-Tuple

Return type Integer

3.1.9 Player

The Player (page 33) class is the parent class of your AI player. It is just like a template. Your AI player class (called PacPlayer) must implement the following methods.

Player.**calculate_direction**(*maze*, *ghosts*, *pac*, *score*)

This is method decides Pacman's moving direction in the next 16 frames (See *Frame Concept* (page 24)). The parameters represent the maze, ghosts, Pacman, and score after the execution of last frame. The game will call this method to set Pacman's direction and let him move (see Pacman's *Move* (page 22)) in that direction for 16 pixels in the next 16 frames.

Parameters

- **maze** – A Maze (page 28) object representing the current maze.
- **ghosts** – An array of Ghost (page 30) objects representing the four ghosts.
- **pac** – A Pac (page 31) object representing Pacman

- **score** – `Integer`. The current score.

Return type `MoveDir` (page 27)

`Player.`**`on_level_start`**(*maze*, *ghosts*, *pac*, *score*)

This method will be called by the game whenever a new level starts. The parameters represent the game objects at their initial states. This method will always be called before `calculate_direction()` (page 33) and `on_new_life()` (page 34).

Parameters

- **maze** – A `Maze` (page 28) object representing the current maze.
- **ghosts** – An array of `Ghost` (page 30) objects representing the four ghosts.
- **pac** – A `Pac` (page 31) object representing Pacman
- **score** – `Integer`. The current score.

Return type `None`

`Player.`**`on_new_life`**(*maze*, *ghosts*, *pac*, *score*)

This method will be called by the game whenever Pacman receives a new life, including the first life. The parameters represent the repositioned game objects. This method will always be called before `calculate_direction()` (page 33) and after `on_level_start()` (page 34).

Parameters

- **maze** – A `Maze` (page 28) object representing the current maze.
- **ghosts** – An array of `Ghost` (page 30) objects representing the four ghosts.
- **pac** – A `Pac` (page 31) object representing Pacman
- **score** – `Integer`. The current score.

Return type `None`

3.2 Java API

Please go here for the documentation of API for Java AI player.

3.3 Make Moves

3.3.1 What happens when your AI makes a move

In the beginning of every turn, the game will ask a move from your AI, and several methods of your AI will be called.

The naming of the actual methods in Java and Python player are different. So the verbose names are used here.

When a new level starts, the following methods will be call in sequence:

1. on level start
2. calculate direction

When a new life starts, the following methods will be called in sequence:

1. on new life
2. calculate direction

When the game first starts, the following methods will be called in sequence:

1. on level start
2. on new life
3. calculate direction

3.3.2 Time-out

There is a time limit to every time your AI making a move. It's 2 seconds. Your AI should finish executing all methods called in any move within the time limit.

Please note this time limit applies to when a new level starts and when a new life starts.

Additional notes: there may be other classes (Java) or modules (Python) in the player directory. You are free to use them as you wish. However, keep in mind that they are not optimized for performance. You may want to write your own classes/modules to organize your code better.

PYTHON MODULE INDEX

g

Ghost, 30
GhostName, 30
GhostState, 30

m

Maze, 28
MazeItem, 28
MoveDir, 27

p

Pac, 31
Player, 33
PyUtil, 32

INDEX

accessible_neighbours() (in module Maze), 29

BLANK (in module MazeItem), 28
Blinky (in module GhostName), 30

calculate_direction() (in module Player), 33
center() (in module Ghost), 30
center() (in module Pac), 31
CHASER (in module GhostState), 30
Clyde (in module GhostName), 30

dir() (in module Ghost), 31
dir() (in module Pac), 31
distance_to_pac() (in module Ghost), 31
DOOR (in module MazeItem), 28
DOT (in module MazeItem), 28
dot_product() (in module PyUtil), 32
dots_count() (in module Maze), 29
DOWN (in module MoveDir), 27

euclidean_distance() (in module Player), 33

FLEE (in module GhostState), 30
frames_till_recover() (in module Ghost), 31
FRIGHTEN (in module GhostState), 30

get_MoveDir() (in module PyUtil), 32
get_perpendiculars() (in module PyUtil), 33
get_pixel_item() (in module Maze), 28
get_power_dots_tiles() (in module Maze), 29
get_tile_item() (in module Maze), 28
get_vector() (in module PyUtil), 32
Ghost (module), 30
GhostName (module), 30
GhostState (module), 30

height() (in module Maze), 29

IN_HOUSE (in module GhostState), 30
Inky (in module GhostName), 30
is_accessible() (in module Maze), 29
is_corner() (in module Maze), 29
is_dead_end() (in module Maze), 30
is_intersection() (in module Maze), 29
is_straight() (in module Maze), 29

LEFT (in module MoveDir), 27

manhattan_distance() (in module Player), 33
Maze (module), 28
MazeItem (module), 28
MoveDir (module), 27

name() (in module Ghost), 31

on_level_start() (in module Player), 34
on_new_life() (in module Player), 34

Pac (module), 31
Pinky (in module GhostName), 30
Player (module), 33
possible_dirs() (in module Pac), 31
POWER_DOT (in module MazeItem), 28
PyUtil (module), 32

RIGHT (in module MoveDir), 27

SCATTER (in module GhostState), 30
state() (in module Ghost), 31

TELEPORT (in module MazeItem), 28
tile() (in module Ghost), 31
tile() (in module Pac), 31
to_2d_list() (in module Maze), 29

UP (in module MoveDir), 27

values() (in module GhostName), 30
values() (in module GhostState), 30
values() (in module MazeItem), 28
values() (in module MoveDir), 27
vector_add() (in module PyUtil), 32
vector_sub() (in module PyUtil), 32

WALL (in module MazeItem), 28
width() (in module Maze), 29