

**p4**  
User Guide

Version 2.0

Author: Peta Masters, s3243186  
Supervisor: Sebastian Sardina

## Contents

About p4 .....	1
Features .....	1
Use of Terms .....	1
Setting Up.....	2
Directory Structure .....	2
Supplied Files .....	2
Getting Started.....	3
User interface.....	5
Toolbar .....	5
Zoombar .....	5
Menus .....	6
Configuration Settings .....	7
Command Line Interface.....	7
Troubleshooting.....	8

## About p4

The Python Path Planning Project (P4) delivers a simulator that lets you load maps, run search algorithms, and – optionally – display their execution in a graphical environment.

### Features

Using p4 you will be able to:

- load any map that conforms to the “gridworld” domain format
- clearly differentiate between different types of terrain, such as swamp or trees
- set your own start position and goal
- execute a search using one of the supplied agents, such as random, A\*, restart weighted A\*, deadline aware search, real-time A\*
- observe the search path the agent takes
- view closed and open lists – or other sets, as returned by any particular agent
- set a delay between steps so you can track the path at whatever speed you prefer
- use onscreen – or keyboard – controls to pause the search and restart it
- run the search one step at a time
- stop a search before the goal has been reached
- reset a stopped search so you can run it again
- set a deadline for a search and watch the timer count down while it’s running
- see the cumulative number of steps taken while the search is running
- see the cumulative cost of the path while the search is running
- get the total cost, number of steps and time taken for any completed or halted search
- zoom in on a map and drag it around to see areas of interest
- reset a map to its original size and position
- write your own config files to search different maps with different settings
- reload a modified config file without exiting from the program
- load a new map, agent or config file from the GUI
- change the start and goal positions from the GUI or randomly reset them from the menu
- specify whether or not diagonal moves are allowed
- specify whether an agent (if it uses heuristics) should use one based on pythagorian square on hypotenuse (euclid) or on vertical plus horizontal distance (manhattan)
- view current settings in a popup window
- obtain map coordinates at the current mouse position
- run searches from the command line – without using the GUI at all
- obtain total cost, number of steps, time remaining and time taken for any command line search
- run multiple searches consecutively in automatic mode by writing your own ‘wrapper’ script
- create your own Agent and see how it performs

### Use of Terms

**Agent** is the entity that suggests the next move in a search. Sometimes referred to as the ‘planner’ or ‘controller’, it doesn’t necessarily have any intelligence beyond the ability to return a random coordinate.

**Domain** refers to the map plus any additional available info about the search environment, such as the cost of traversing a particular type of terrain, and the size of the map.

## Setting Up

P4 has been tested on Windows 7, Windows 8, Mac OSX and Linux Mint 17. You'll need a system capable of running a Windows or X-Windows-type GUI, with Python 2.7 installed, downloadable from [www.python.org/getit/](http://www.python.org/getit/)

### Directory Structure

P4 expects the following directory structure.

```
maps
src
  |_agents
```

It will search for map files in "maps" and agent files in "agents" – although you can load maps and agents from any directory via the GUI once the program is running. All other source code should be located in "src".

### Supplied Files

You need at least one map and one agent to perform a search.

### Maps

Maps are in "gridworld" file format: every square/pixel of the map grid is represented by an ASCII character and each character equates to a particular terrain type – e.g. S = Swamp, T = Trees, etc.

### Agents

Agents are written as Python modules. Each agent module defines a class "Agent" that conforms to the prescribed interface (see Appendix).

### src

Required P4 program files are as follows.

```
config.py          //default config file, names map and agent
p4.py              //script to start program
p4_controller.py   //module: simulator
p4_model.py        //module: logical map representation
p4_view.py         //module: GUI
p4_view_map.py     //visual map representation
p4_utils.py        //additional required classes
```

You may find additional files in the src directory with the extension 'pyc'. They're compiled files that Python creates to speed up the loading process. You can keep or delete them; they have no impact on the speed of the program or how long it takes a search to reach goal.

## Getting Started

To run a default search (using the agent and map defined in config.py):

1. With Windows or a Windows-like GUI running, open a terminal window.
2. Navigate to the src directory.
3. At the command line, enter:

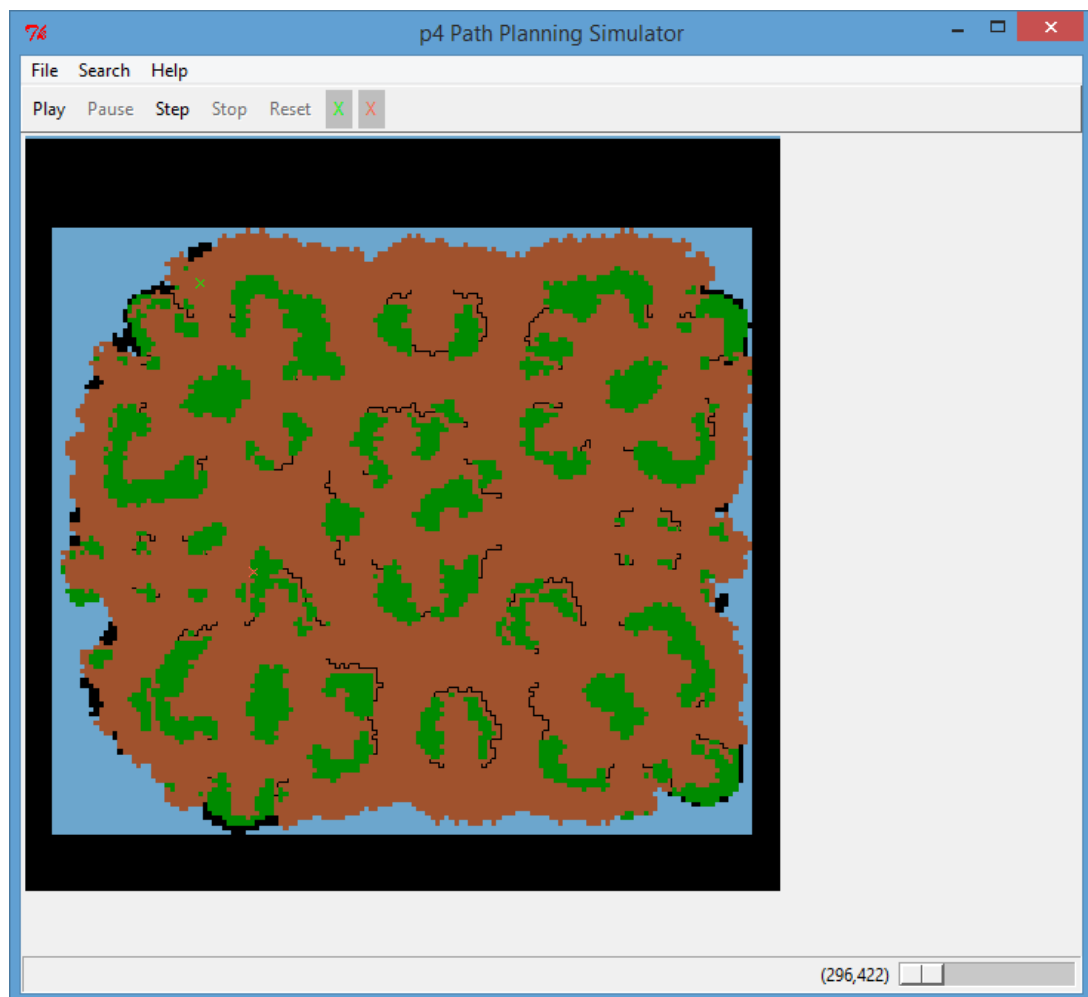
```
python p4.py
```

The simulator initialises, using default configuration settings, and reports progress in the Terminal window.

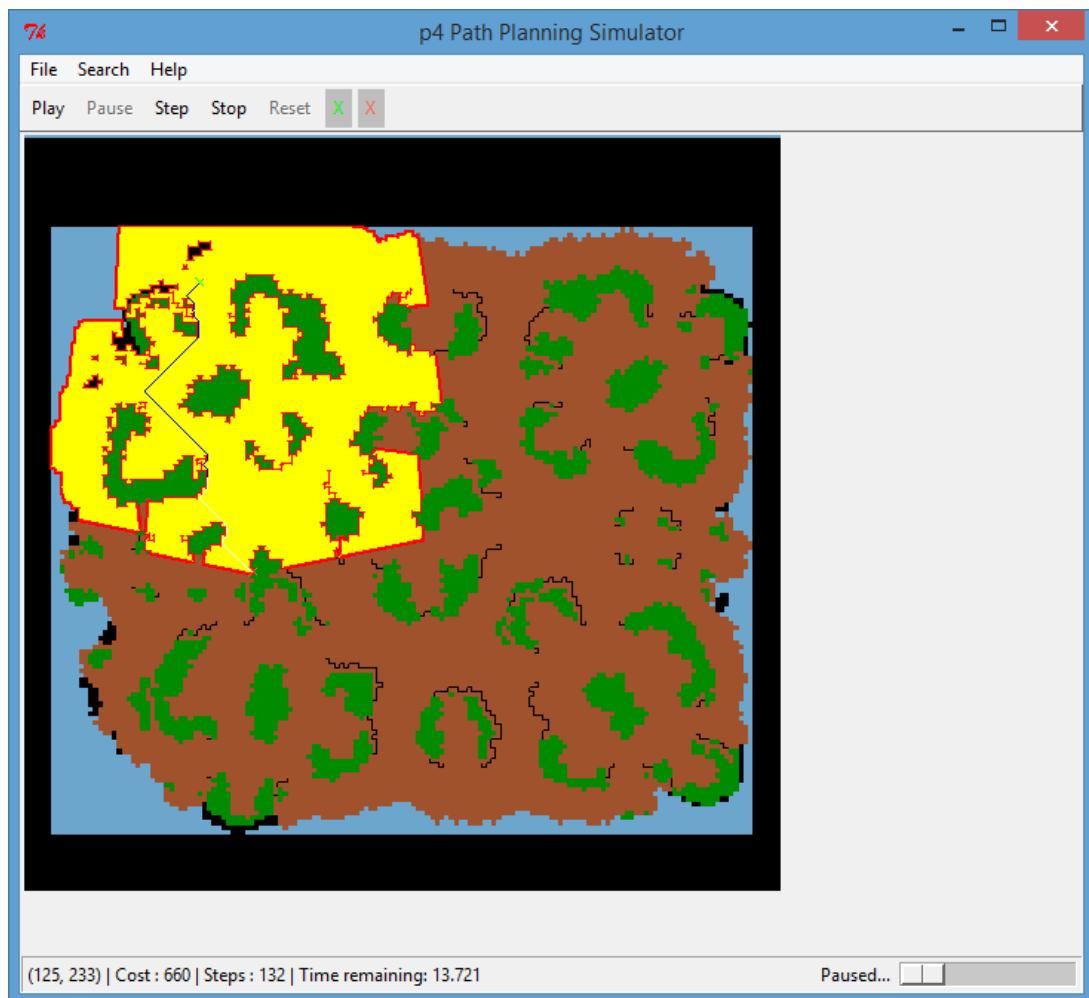
```
Checking for configuration file config.py
initialising simcontroller

Reading config file
Initialised agent_astar
Launching GUI...
```

4. After a few moments, the GUI opens with map loaded.



5. To start – or pause – a search, hit “S” on the keyboard or click Play (or Pause).



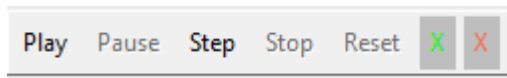
6. When the goal has been found, the number of steps, total cost, final time remaining, and total time taken appear on the status bar.





## User interface

### Toolbar

Use the buttons to control the search.



BUTTON	Quick Key	ACTION
Play	S	Start searching from the start position or to restart after a pause.
Pause	S	Pause a started search.
Step		Move one step, then pause.
Stop		Permanently terminate a paused search.
Reset		Remove the search path from the map, reset the step and cost counters to zero, and restore the full time allocation.
		Reposition the start point. Click the green cross, then click the position on the map where you want the start point moved to.
		Reposition the goal. Click the red cross, then click the position on the map where you want the goal moved to.

### Zoombar

To zoom the map to up to 16x its original size.



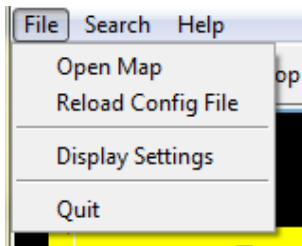
Drag the slider (bottom-right of the screen) to the right. Click and drag anywhere on the map to drag it around so you can see the area you're interested in.



Click the Reset button to restore the map to its original size and position.

## Menus

You can check and modify settings using the menu.

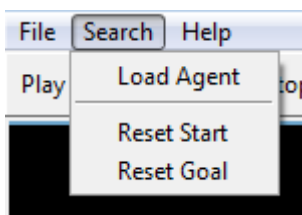


To open a different map, select Open Map from the File menu and navigate to the map you want. Be aware that the bigger a map is, the longer it will take to load. Start and goal positions will be randomly allocated.

To change configuration settings, such as DIAGONAL or DEADLINE, modify and resave the config file, then select Reload Config File from the File menu.

If you have made changes since loading the config file and wish to check the current configuration, select Display Settings from the File menu.

To quit the program, select Quit from the File menu, close the window or, at the command line, hit Ctrl-C.

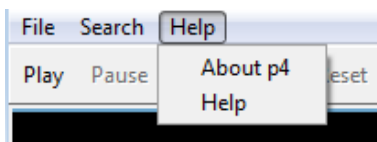


To load a different agent, select Load Agent from the Search menu and navigate to the agent you want.

To move the start or goal without using the toolbar, choose Reset Start or Reset Goal from the Search menu. Onscreen green and red crosses mark the new – randomly allocated – locations. Check the foot of the screen for the specific coordinates.



Alternatively, hover your mouse over any part of the map and the screen coordinates for that location will be displayed just left of the zoombar.



To check the p4 version you are running, select About p4 from the Help menu.

To remind yourself of the above functionality, select Help.



## Configuration Settings

You can modify configuration settings to open any properly formatted map, set any valid start and goal coordinates, and nominate any correctly formed search agent.

The default config file, config.py, contains settings similar to the following.

```
AGENT_FILE = "agent_das.py " #agent filename - must be in src/agents

MAP_FILE = "blasted.map"      #map filename - must be in maps directory
START = (120,101)             #coordinates of start location
GOAL = (220,301)              #coordinates of goal location

GUI = True                    #True = show GUI, False = run on command line
SPEED = 0.0                   #delay between displayed moves, in seconds
DEADLINE = 4                  #number of seconds to reach goal
HEURISTIC = 'euclid'          #may be 'euclid' or 'manhattan'
DIAGONAL = True               #True = allow diagonal path, False = disallow
```

To run the simulator using your own configuration file, at the command line, enter

```
python p4.py <config_file>
```

AGENT\_FILE must be set to the filename of a valid Python module containing an Agent class. The file is assumed to reside in the agents directory.

MAP\_FILE must be set to the filename of a mapfile in movingai benchmark format. The file is assumed to reside in the maps directory.

START and GOAL should be valid map coordinate. If the specified coordinate is not traversable, it will be replaced by the closest traversable alternative. Note that the brackets around the coordinates are required.

If you set GUI = False, the search will run 'invisibly' – and more quickly – before displaying the result.

If you set a value for DEADLINE, an onscreen timer will count down from the time you specify. If the agent fails to complete the path within the deadline, its search will be terminated. On Unix or Linux systems an additional timeout feature is activated: if an Agent fails to return altogether and *twice* the time allowed by the DEADLINE passes, a timeout occurs, returning control to the system.

## Command Line Interface

You can run p4 without a GUI. If you specify all options on the command line, p4 will return the result to the command line. (Note that Windows requires brackets around the coordinates, but Linux – below – does not).

```
mint@mint ~/Share/p4-simulator/src $ python p4.py -m blasted.map -s 245,233 -g 322,120 -a agent_astar -e manhattan -d 1 -nodia
Initialised agent_astar
Searching...
Total Cost : 945 | Total Steps : 189 | Time Remaining : 0.65689 | Total Time : 0.34311
```

For details of all the available switches and their meanings, see the README file in p4's top level directory.

## Troubleshooting

The configuration file can have any name but needs to contain a valid list of Python variables. If there seem to be problems with the system not 'remembering' settings, check the following.

- A hash symbol (#) signals a comment in Python. Check something important hasn't been commented out.
- Check that, in editing the config file, multiple values haven't been entered for the same variable.
- Check that True and False are correctly capitalised in the GUI setting.

If the GUI loads without a map:

- Check the map named in the config file exists in the maps directory and that it hasn't become corrupted.

To remind yourself of p4 options, navigate to the src directory and, at the command line, enter:

```
python p4.py -h
```

## Write Your Own Agent

The purpose of the Agent is to tell the SimController what move it should make next. The decision may be random, may be based on a well known algorithm, may take into account the settings requested by the user (in the config file) or may ignore them.

To work with p4, the agent file should be created as Python module with .py extension and saved to src/agents. The module must contain a class called Agent.

The Agent constructor will be passed all settings from the configuration file, plus a reference to the LogicalMap object, which the Agent will need to interrogate in order to figure out where to move next.

To allow for future expansion, the constructor should be able to accept *any* keyword arguments passed in:

```
__init__(self, **kwargs)
```

Agent must support two functions: getNext() which returns the next move and reset(), which the SimController will call if config settings change.

```
getNext(self, mapref, current, goal, timerremaining)
```

where:

- `mapref` is a reference to a LogicalMap object, which you can interrogate about the domain (see below)
- `current` is the current map coordinate formatted as a tuple, (col,row)
- `goal` is the target coordinate formatted as a tuple, (col,row)
- `timerremaining` is the number of seconds before the configured deadline will be reached

Must return a valid next move as a coordinate (col, row).

Optionally, getNext() can also return sets of coordinates to be displayed in a particular colour or to be reset to the original map terrain. These sets of coordinates should be returned in the format (list, color) where list is a list of coordinates (e.g. [(200,300),(201,300),(433,400)] ) and color is a color name such as "purple", "white", or (using hex values) "#40E0D0".

**Note:** a valid move is any passable coordinate adjacent to the current coordinate. Beware that p4 will cost your path as 'infinite' if you attempt to cut corners past an obstacle: such a move is regarded as impassable. For a full list of valid color names, refer to <http://wiki.tcl.tk/16166>

```
reset(self, **kwargs)
```

where:

- `kwargs` may include any settings from the config file, plus a reference to the LogicalMap object – exactly as described for the constructor, above.

## LogicalMap Interface

Use the reference passed to the Agent in `mapref` – or in `kwargs["MAPREF"]` – to interrogate the current LogicalMap object for information about the domain.

```
getCell(coord)
```

returns the content of the cell at that coordinate, i.e. an ASCII character, one of .@GOSTW

```
getCost(coord)
```

returns the cost of the terrain type at that coordinate

`getWidth()`

returns the width of the map, based on the number of characters in its first row

`getHeight()`

returns the height of the map, based on the number of rows

`isAdjacent(coord_a, coord_b)`

returns True if coord\_a is adjacent to coord\_b and False otherwise. Adjacent may mean horizontally, vertically, or diagonally adjacent or horizontally and vertically adjacent only – depending on whether the user has set DIAGONAL to True or False in the config file.

`isPassable(coord, [parent])`

returns True if the terrain at coord is passable, and False otherwise. Use the optional parent parameter to check if a coordinate is passable if accessed from the parent location (e.g. to check that you won't be cutting a corner).

`getAdjacents(coord)`

returns list of passable coord tuples that are horizontally, vertically, or diagonally adjacent to the coord passed in. See isAdjacent() for explanation of 'Adjacent'.

`getAllAdjacents(coord)`

returns list of all coord tuples, whether or not they are passable, horizontally, vertically or diagonally adjacent to coord – regardless of config file setting for DIAGONAL.

`getH(coord_a, coord_b)`

returns the minimum distance between coord\_a and coord\_b, based on either manhattan or euclidian calculations depending on the setting for HEURISTIC in the config file.

In all cases where a coord is passed in, it should be formatted as a tuple (col,row). Note that this usually means passing parameters inside double brackets, e.g:

```
stepCost = mapref.getCost((100,200))
```

For additional functions, check out the source code (p4\_model.py) and read the scripts in the agents directory to see what they do – and how they do it.