COSC 2409 Programming Project 2

Path Planning Simulator
Project Documentation

Developed by
Peta Masters
s3243186

Supervisor
Sebastian Sardina

Draft Deadline: 25 October

Final Deadline: 11 November

The Python Path Planning Project (p4) was undertaken as a Programming Project for COSC2409 in Semester 2, 2013.

The p4 Project documentation comprises:

- **Executive Summary**
- **User Guide**
    - Appendix: Write your own Agent
- **Developers' Guide**
    - Appendix A: Requirements / Roadmap
    - Appendix B: Map Format
- **Test Scripts**
- **Presentation Slides**
- **Project Log**
- **Project Plan** (as submitted previously)
  – includes schedule

Executive Summary

The brief:

to develop a path planning simulator in which it is possible to load different maps in "gridworld" format, run various search algorithms, and display their execution in a graphical environment.

p4 enables users to:

- load into a GUI interface any map that conforms to the "gridworld" domain format
- clearly differentiate between different types of terrain, such as swamp or trees
- set their own start position and goal
- execute a search using one of the supplied agents – either random or A* – and observe the path it takes
- set a delay between steps to track the path at their preferred speed
- 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 to 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
- write their own config files to search different maps with different settings
- run searches from the command line – without using the GUI at all
- obtain total cost, number of steps, and time taken for any command line search
- create their own search agents and see how they perform

Design elements:

- p4 can run with or without the GUI
- any agent module can be substituted provided it conforms to the supplied API
- use of argparse in the launch script has opened the possibility of adding configuration options to the command line
- the LogicalMap interface is fully defined: can be tested independently and could be replaced
- the search function uses a next step generator to display and cost each route segment
- GUI search is implemented using a separate generator that repeatedly 'yields' control back to the GUI, making the search/step/pause mechanism particularly simple and robust

Limitations:

- zoom is not fully functional – more a proof of concept than a feature
- reset doesn't reposition a map that has been moved or zoomed
- most features marked as 'Could-Do' in the requirements are not yet - or only partially - implemented, e.g. modifying terrain from the GUI
- incomplete validation: the agent's next move is not validated and neither are features that will eventually be available via the GUI, e.g. setting an impassable goal coordinate
- the map's 'viewport' is too small: it is currently determined according to map size rather than the available width/height of the user window and is not resizable
- GUI load time currently restricts the map size to significantly less than the 10,000 x 10,000 pixels demanded by non-functional requirements

USER GUIDE

Appendix A: Write your own Agent

p4
User Guide

Version 1

Author: Peta Masters, s3243186
Supervisor: Sebastian Sardina

Contents

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 – to conduct either a random or A* search
- observe the search path the agent takes
- 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*
- write your own config files to search different maps with different settings
- run searches from the command line – without using the GUI at all
- obtain total cost, number of steps, and time taken for any command line search
- create your own Agent and see how it performs

** **Note**: you can't currently search a map once it's been zoomed or moved, and the behaviour of a zoomed search path is undefined.*

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.

System Requirements

P4 has been tested on Linux and Windows 7. 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/

Directory Structure

P4 expects the following directory structure.

```
maps
src
   |_agents
```

It will search for map files in "maps" and agent files in "agents". 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

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 and visual map representation
```

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.

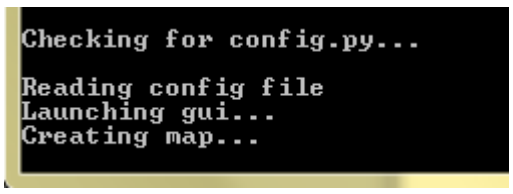
Getting Started

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

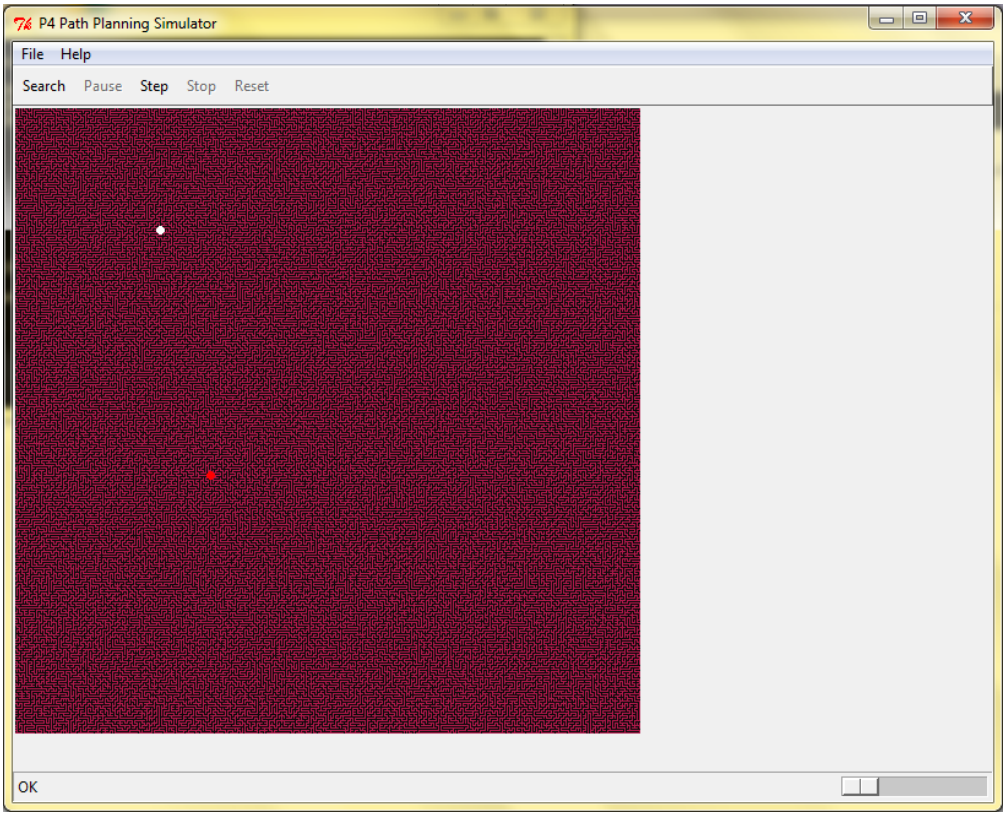
1. With Windows or a Windows-like GUI running, open a terminal window.
2. Navigate to the src directory.
3. At the command line, enter:

```
python p4.py
```

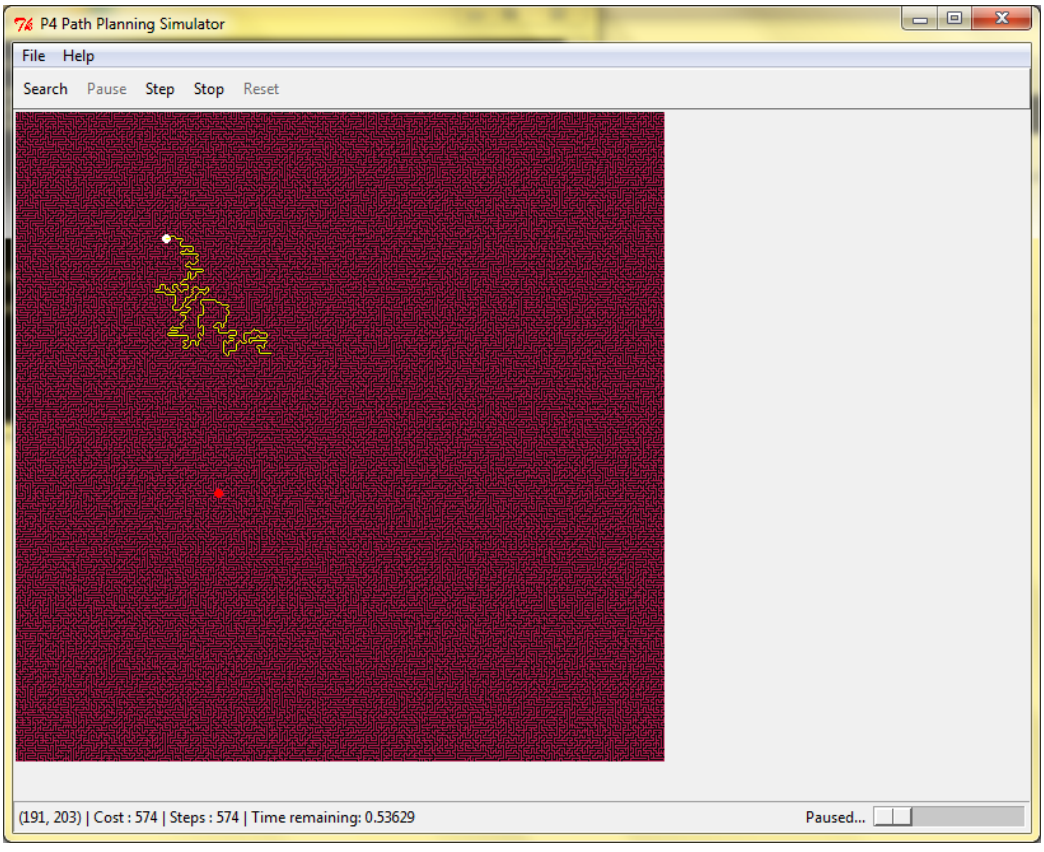
The simulator initialises, using default configuration settings, and reports progress in the Terminal window.



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



5. To start – or pause – the default search, hit “S” on the keyboard or click Search.



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

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 is config.py, which, as supplied, contains the following settings.

```
AGENT_FILE = "agent_astar.py" #agent file name

MAP_FILE = "blasted.map"      #map file name
START = (100,194)             #coordinates of start location
GOAL = (110,204)              #coordinates of goal location

GUI = True                    #True = show GUI, False = run on command line
SPEED = 0.0                   #delay between moves in seconds
DEADLINE = 2                  #number of seconds to reach goal
```

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 coordinates of the nominated map. Both coordinates should point to traversable locations on the grid. 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 SPEED, there will be a delay – of whatever length you've nominated – before each move is displayed on screen.

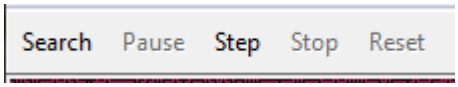
If you set a value for DEADLINE, an onscreen timer will count down from the time you specify so you can easily see whether that deadline is exceeded.

***Note**: the speed and deadline settings have effect only if GUI is set to True and are otherwise ignored.*

User interface

Toolbar

Use the buttons to control the search.



BUTTON	Quick Key	ACTION
Search	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.

Menu

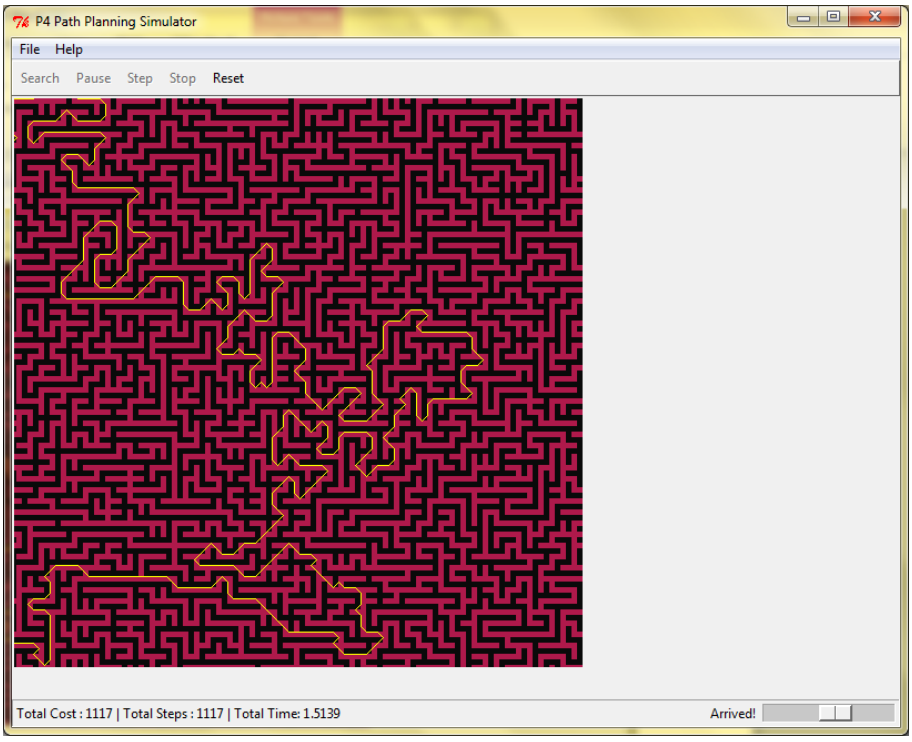
To open a different map, select Open Map from the File menu and navigate to the map you want. Be aware that the bigger a map is, the longer it will take to load.

***Note**: you cannot currently search a map opened via the menu, because no mechanism has been provided for you to set new start and goal coordinates.*

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

Zoombar

To zoom the map to up to 8x its original size, drag the slider (bottom-right of the screen) to the right. Click and drag anywhere on the map to move it around so you can see the area you're interested in.



***Note**: zoom cannot yet be used to observe a search while it is executing.*

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

A p4 agent file should be created as Python module with .py extension and saved to src/agents. The module must contain a class called Agent which supports two functions:

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

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, (y, x)
- `goal` is the target coordinate formatted as a tuple, (y, x)

Returns a valid next move as a coordinate (y, x).

***Note**: a valid move is any passable coordinate adjacent to the current coordinate.*

`reset(self, mapref, current, goal)`

where arguments are as defined for getNext().

***Note**: The Agent can assume that the Simulator will call reset() if any of the three parameters have changed since the last getNext request, e.g. if the Reset button has been clicked or – in future – if the map or goal has changed so that the path needs to be recalculated.*

LogicalMap Interface

Use the reference passed to the Agent in 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 – whether horizontally, vertically, or diagonally – and False otherwise

`isPassable(coord)`

returns True if the terrain at coord is passable, and False otherwise

`getAdjacents(coord)`

returns list of passable coord tuples that are horizontally, vertically, or diagonally adjacent to the coord passed in

In all cases where a coord is passed in, it should be formatted as a tuple (y, x) where y is the vertical coordinate (row) and x is the horizontal coordinate (column). Note the double brackets, e.g:

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

DEVELOPERS' GUIDE

Appendix A: Requirements / Roadmap

Appendix B: Map Format

p4
Developers' Guide

Version 1

Author: Peta Masters, s3243186
Supervisor: Sebastian Sardina

Contents

About this document

The purpose of this document is to provide future developers working on the Python Path Planning Project (p4) with a clear understanding of its design and intended functionality.

This document also sets out the limitations of the project to date: it lists known bugs, and describes proposed enhancements.

Audience

It is assumed that readers of this document are familiar with Python and have already read the p4 User Guide.

Project Overview

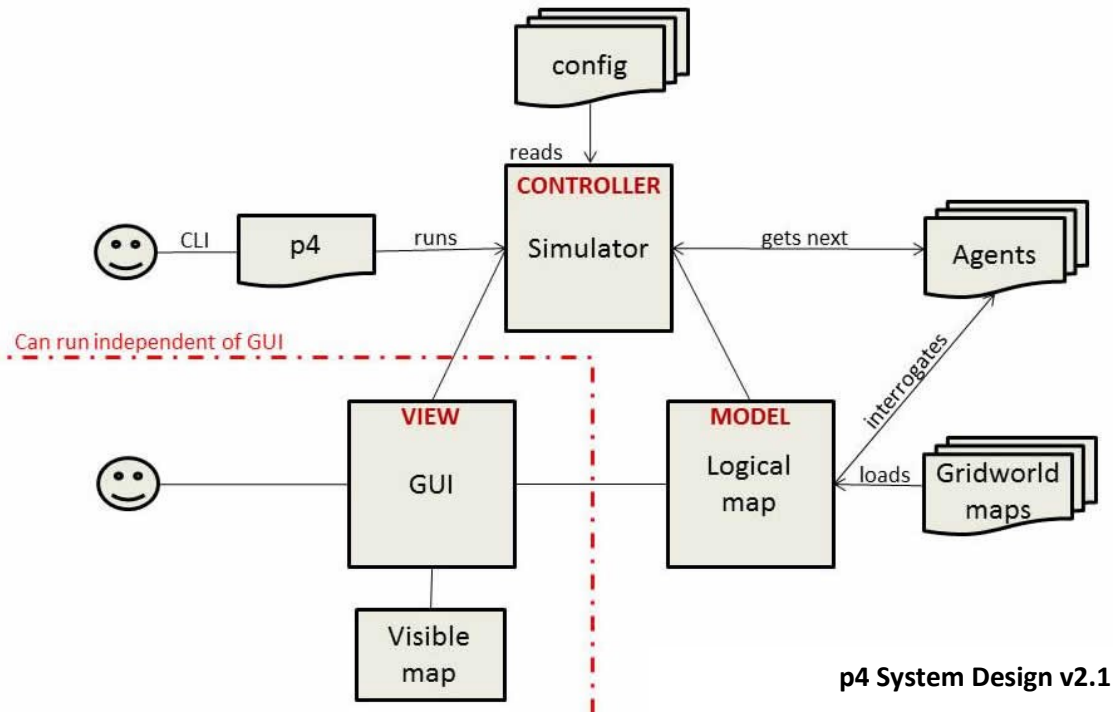
Undertaken as an undergraduate Programming Project for COSC2409 in Semester 2, 2013, the brief was to build "a path planning simulator in which it is possible to load different types of maps, run various search algorithms, and display execution in a graphical environment. The end result should be similar to Apparate (see links) but based on Python."

Requirements for p4 were based on observation of the running application and examination of the code. They were framed as user stories and were repeatedly revisited and reprioritised as the project progressed. The current requirements spec now provides a record of what's built, what's partially built, and what's still to do (see Appendix A).

The requirements spec also includes non-functional requirements, such as the need for extensibility, which should be carried forward into future development.

System Design

It's an MVC architecture, with 4 core classes, plus agents, maps, startup script, and config files.



p4 System Design v2.1

***Note**: in the planning domain, 'controller' is used synonymously with 'agent' so 'Simulator' is preferred when talking to SMEs.*

Components

p4	Startup script
config	Config file for user configurable settings – e.g. map and agent filenames
Simulator	Core component orchestrates the rest
Agent	Given map reference, current position and goal, returns the next move
Map	Ascii text file represents map in Gridworld format
LogicalMap	Representation of the Gridworld map that can be interrogated by other components
GUI	A discrete element – p4 can run searches without a GUI
VisibleMap	Visual representation of the Logical Map, as it appears within the GUI

The code itself is fully commented and can be interrogated using help(). However, the following sections provide additional information, particularly in relation to design decisions and proposed development.

p4.py

Launches P4 by initialising the Simulator with the name of a config file.

- Uses argparse to accept the config filename as a command line argument.
- Checks that the config file exists. If not, displays an error and exits.
- If there are no command line arguments, passes in "config.py" as a hardcoded default.

Why argparse?

Using argparse provides usability/error-handling functionality for free, e.g –h to display usage and info, error message for unrecognised switches, error message for too many arguments

It also provides easy extensibility options – anything currently set in the config file could in future be passed in via argparse (e.g. agent filename, map filename, etc).

config.py

Config file parameters and usage is set out in the User Guide. Note also:

- The config file is a text file and can have any name (doesn't need .py extension) but must contain a valid list of Python variables.
- Simulator reads the config file straight into its config dictionary variable.
- Some validation is performed while the file is being read in – but it's not exhaustive. If validation fails, the controller displays an error message and terminates the program.

Why this file format?

I used an ordinary list of Python variable assignments for simplicity and speed and because it can be extended so easily: just add whichever new variables you need and you can retrieve them from Simulator.config without any extra programming effort. Provide default values for new variables to avoid having to apply any version control for config files – defaults can keep it all backward compatible.

Simulator

- Reads in the config file and initialises Agent and LogicalMap
- If GUI = False in config, performs search and reports result to command line
- If GUI = True, imports view module, initialises GUI, and waits for user action.

- Currently, also initialises VisibleMap object and subsequently interacts with it directly – a design decision that may need revisiting (see below).

Simulator handles search by means of a generator that requests one step at a time from the Agent. The same generator is used whether search is performed via the GUI or command line and whether it is running as a continuous search or step by step.

Simulator tracks path cost, total steps and time, and relays this information back to the GUI for display. It has menu handlers and button handlers which provide an interface to the GUI.

The loadmap function is only partially realised and marked TODO.

Agent
Details of the Agent interface requirements are supplied in an Appendix to the User Guide.

Map
For details see Appendix B.

LogicalMap
See Appendix to User Guide for interface presented to the Agent. Note also:

- LogicalMap reads the Gridworld map data directly into a matrix (list of lists). It reads the preamble (width, height, and costs) into a costs dictionary variable.
- In order to create the costs dictionary correctly it's initialised as abbreviation: term (i.e "G": "ground", etc.) and then the costs that are read in from the map are switched with the term that they match. Costs can then be read back based on the actual content of any given cell (e.g. "G": 1, "S":10, etc).
- If a value is given as infinity (+inf) in the map preamble, it's stored in the LogicalMap costs dictionary as None – i.e. a cell whose cost == None is impassable.
- If an attempt is made to access a non-existent cell (e.g. 500,500 in a 300 x 300 mapgrid), LogicalMap flags the error by printing the requested cell coordinates to the console, but treats the cell as 'impassable' and attempts to proceed.

GUI
The decision was made to use Tkinter (Python's native GUI) at the design stage.

- I had planned to try switching in different GUIs and see if any of them have particular advantages.
- Tkinter is fiddly but well documented and it's easy to tap into considerable expertise on sites such as stackoverflow. T
- The zoom is yet to be properly implemented. The current partial implementation suggests it will be possible – but it may be worth looking at the support given to this sort of functionality by alternatives.

VisibleMap
Experimentation is needed with the VisibleMap in particular to assess exactly which sort of object it should be and also how tags can best be used to achieve the layering and zooming effects that we need. Currently:

- The VisibleMap object contains a frame, which contains a map canvas (see Limitations below).
- The map itself is created as a PhotoImage object - for which, limited documentation is available. The good thing about the PhotoImage is that it provides very immediate access for

the literal 'bitmapping' necessary to translate the gridworld characters (now held in the LogicalMap object) into an image.

- PhotoImage is added to the canvas as an image item. The search path and the start and end points are ordinary canvas items. Currently, they are tagged and the tags are used to remove the path when it is reset.

Event-handling is used to manage the click and drag mechanism and also to register the keyboard events. I have only implemented one 'toggle' keypress ("S" to search/pause). It will be a simple extension to add any additional keyboard activation, following the same model.

Known Bugs

#	Description	Status
1	Status bar timer is only showing to 3 decimal places in Linux – it's set to show to 5, and does show 5 under Windows.	
2	Zoom is only partially functional – more a proof of concept than a feature.	
3	Bug or 'feature'? - It's possible to restart a stopped search even though buttons are inactive by pressing the 'S' key.	

Limitations/Proposed Enhancements

Start/Goal

- Can't set/change start or goal from GUI
- Doesn't validate start or goal settings – behaviour if you set a goal that can't be reached is undefined.

  [Ideally, system should identify closest traversable coordinate and prompt user to confirm the revised coordinate is OK or reenter. If setting by click and drag, should only permit drop on traversable coordinate.]

Reset

- If map has been zoomed or dragged, reset should reset it to its original zoom and position.

Search

- Simulator does not validate the Agent's move to make sure it's legal.
- The LogicalMap.getAdjacents() method returns all adjacents including those on the diagonal – there's no option to return only those coordinates that are vertically and horizontally adjacent.

ViisibleMap

- Simulator currently accesses the VisibleMap directly instead of going through the GUI. This was a deliberate decision – to minimise time taken to draw the map – but it means that the map can't be completely isolated for testing. Raises question/decision to be made: encapsulation versus speed?
- The VisibleMap contains a frame which contains the canvas. it might be better – and more correct – for the GUI to contain the frame, and for the VisibleMap object to be derived directly from Tkinter.Canvas (not from 'object').

- I have implemented a very simple click and drag mechanism to move the map around within the frame – but currently this is really only a proof of concept.

- When the functionality is added so the user can change the terrain of the current map, I envisage a row of tool buttons, each with a single letter in the colour being used for the terrain type. If the user clicks that button or presses that key, the button takes on a 'depressed' state and anywhere you click on the map will be coloured with the related colour (and the corresponding terrain type will be copied to the logical map).

Zoom

- The frame in which canvas is displayed (effectively the 'viewport') is set according to the width and height of the map when it's loaded – but it should be set to the available width/height of the user window and should be sizeable by the user and/or resized when the user resizes the window. Currently, if you load a tiny map, you get a tiny viewport – so you can't zoom in on it even though there's plenty of screen real estate available.

See Appendix for list of specified requirements not yet – or only partially – implemented.

Useful Links

Apparate: https://sites.google.com/site/ssardina/research/apparate-path-planning-simulator

Python official documentation: http://docs.python.org/2/

Tcl/Tk Manual Pages: http://www.tcl.tk/man/

Tkinter API documentation: http://mgltools.scripps.edu/api/DejaVu/Tkinter-module.html

Tkinter Book: http://effbot.org/tkinterbook/

Introduction

p4 requirements were initially based on observation and reverse engineering of Apparate, but should continue to be modified and reprioritised as the project progresses.

***Note**: this is a living document, intended to be maintained throughout the life of the project.*

Requirements

Requirements are represented as user stories and prioritised on the Moscow model: (M=Must, S =Should, C=Could, W=Won't).

- Stories highlighted have been fully implemented.
- Stories highlighted have been partially implemented.
- Non-functional requirements are ongoing and therefore un-highlighted – but note that currently NF8 has **not** been achieved.

#	Requirement	Priority
1	Read in any available map supplied in 'gridworld domain' format	M
2	Read in any available map supplied in Apparate format (i.e. including costs for semi-navigable cells such as water, swamp, etc.)	M
3	Display default blank map	M
4	Display map, distinguishing between different features of the mapped terrain	M
5	Optionally, display 'traversable' map, distinguishing only between navigable/ unnavigable areas	C
6	Select between and load any available map from GUI	C
7	Add water to current map	C
8	Add swamp to current map	C
9	Add trees to current map	C
10	Add ground to current map	C
11	Add out-of-bounds to current map	C
12	Add start-point to current map	C
13	Add goal to current map	C
14	Resize map (i.e. resize by dragging interface)	W
15	Zoom in	S
16	Zoom out	S
17	Zoom to selected area (i.e. so that selected area fits bounding box))	C
18	Move map within bounding box window	S
19	Build map, based on script	W
20	Reset to display most recently loaded map	S

#	Requirement	Priority
21	Run a default search (which may be random or A*) by passing map, current position and goal to planner – planner returns new grid position (n.b. requires simulator/planner interface design)	M
22	Run a meta A* search – i.e. using a meta-heuristic TBD	C
23	Select between multiple search algorithms/planners from GUI	C
24	Specify whether the search is allowed to use diagonal movement (Euclidian) or vertical/horizontal (Manhattan) only. n.b. may require interface with rules module to determine ismovelegal(curr,nxt)	C
25	Display search path on map from start point to goal	M
26	'Animate' the movement of a search agent following the path in discrete observable steps	M
27	Optionally, pause for 1 second at each step (before getting next step)	M
28	Optionally, pause display and Next button after each step then wait for user to click.	S
29	Optionally, display path travelled so far *Dev note: currently path is **always** displayed – i.e. it's not optional*	M
30	Optionally, draw/fill area searched (i.e. expanded nodes) n.b. requires inclusion of callback function in P4/planner interface.	S/C
31	Optionally, draw frontier of search (i.e. unexpanded nodes) n.b. requires inclusion of callback function in P4/planner interface	S/C
32	Include callback function in P4/planner interface to draw path (typically from current position to goal)	S
33	Optionally, halt the search	M
34	Display status bar, showing the current step and whether it is running, pausing, stopped or in an error state	S
35	Show grid coordinates of current mouse position	C
36	Indicate currently active function/mode, if applicable	C
37	Permit zoom to be modified interactively while the program is running.	S
38	Permit map terrain (requirements #7-13) to be modified interactively while the program is running	C
39	Permit search path animation options (requirements 26-30) to be modified interactively while program is running	W
40	Permit search algorithm/controller to be switched (requirement 23) while program is running	W
41	Permit all above 'live' changes (requirements 35-53 to be actioned via a script.	W
42	Display search path calculation time (i.e. planner time only) on current processor for most recently completed search on status bar	M
43	Display or log number of base programming steps executed for last search	W
44	Display cost of path for most recently completed search on status bar	M
45	Display search paths generated by multiple algorithms overlaid on one another in different colours to facilitate comparison.	W
46	Perform the search without displaying the GUI	M

#	Requirement	Priority
47	Return statistics about the mat programmatically - e.g. percentage of map covered by water	S
Constraints and Non-Functional Requirements		
NF1	The project is to be implemented within 10 weeks by one developer, working part time.	M
NF2	Program design must be extensible.	M
NF3	The code must be especially clean and well-commented, and documentation must be complete on handover, so that the project can readily be completed/extended by any other Python programmer.	M
NF4	Whilst preferring an attractive interface, accessibility and ease of use are to be favoured over look and feel.	M
NF5	The system should provide feedback to the user (e.g. via a status bar), so they know the effect of any key presses or mouse movements and are always aware of the current status of the search.	S
NF6	Deliverable must be able to run on Linux	M
NF7	Must be possible to set options (e.g. map to load, planner to use, speed and zoom) from the command line	M
NF8	Must be able to support interrogation of maps up to 10,000 x 10,000 without any observable drop-off in performance *Dev note: This requirement is unmet. Even maps of 1000 x 1000 are painfully slow to load.*	S

Map Format

The P4 map format is as for Appartate, which is adapted from Nathan Sturtevant's movingai gridplan format (Sturtevant, monivingai.com).

In common with the gridplan format, data is stored as an ASCII grid and may include any of the following characters:

```
. - passable terrain
G - passable terrain
@ - out of bounds
O - out of bounds
T - tree (unpassable)
S - swamp (passable from regular terrain)
W - water (traversable, but not passable from terrain)
```

All map files have a preamble, which begins with the lines:

```
type octile
height x
width y
```

where x and y are the respective height and width of the map.

Files in Appartate map format may then list the map's terrain types with their traversal costs, e.g:

```
ground 5
tree +inf
swamp 10
water 20
```

Finally, the preamble to all maps ends with the single word:

```
map
```

which is immediately followed by *heght* x *width* rows of map data.

0,0 always represents the top left-hand corner of the map.

***Note**: Sturtevant advises that, if the number of characters in the ASCII data differs from the height and width supplied, the map should be scaled to match – but in P4 any discrepancy results in an error condition.*

Example

```
type octile
height 9
width 75
map
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWGGGGGGGGGTTTTTTTTGGGGGGGGSSSWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWGGGGGGGTTTTTTTTGGGGGGGGSSWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGSSWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWGGGGGGGGGWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
```

References

Sturtevant, N, *Pathfinding Benchmarks: File Formats*, movingai.com, last accessed 18/10/13 from http://movingai.com/benchmarks/formats.html

TEST SCRIPTS

General Preconditions

- src directory content, maps, and agents set up as described in user guide.

Startup Script (p4.py)

Test #	Test	Actions	Expected output	Pass /Fail
1	Execute with default config file	python p4.py	Successfully loads GUI and map	P
2	Default config file not present	Pre: rename config.py to xconfig.py python p4.py	Error message identifies unable to locate config file	P
3	Execute with named config file	python p4.py xconfig.py Post: rename xconfig.py back to config.py	Successfully loads GUI and map	P
3	Named config file not present	python p4.py numpty	Error message identifies unable to locate config file	P
4	argparse help	python p4.py –h	Displays usage	P
5	argparse with unknown option	python p4.py –g config.py	Displays usage	P

Config file (config.py)

- ensure tests directory and test config files 1-7 installed

Test #	Test	Actions	Expected output	Pass /Fail
1	test alt agent	python p4.py tests/test1.cfg	Runs with random agent (may not get far!)	P
2	test alt map	python p4.py tests/test2.cfg	Runs with A star agent on “blasted” map, not default maze	P
3	test alt start/goal cords	python p4.py tests/test3.cfg	Runs with A star agent on “blasted” map, searching east to west	P
4	test without deadline	python p4.py tests/test4.cfg	Displays as for test 4 but without displaying countdown	P
5	test with alt deadline and delay	python p4.py tests/test5.cfg	displays as for 4 but with ¼ sec delay between moves	P
6	test without GUI	python p4.py tests/test6.cfg	displays results as for 4, but on command line	P
7	test with random missing parameters	python p4.py tests/test7.cfg	displays default map – search unavailable	P

LogicalMap (model.py)

Strategy: test the model module by loading it into the interpreter and interrogating the interface.

Test #	Test	Actions	Expected output	Pass /Fail
prep	Run interpreter import model create logical map	python import p4_model as m l=m.LogicalMap("../maps/blasted.map")		
1	getCell	l.getCell((1,1)) l.getCell((4,1)) l.getCell((600,600))	'W' '@' row:600 col:600 '@'	P
2	getCost	l.getCost((1,1)) l.getCost(4,1)) l.getCost((600,600)) type(l.getCost((4,1)))	20 [no response] row:600 col 600 (type 'NoneType')	P
3	getWidth()	l.getWidth()	512	P
4	getHeight()	l.getHeight()	512	P
5	isAdjacent()	l.isAdjacent((5,5),(5,6)) l.isAdjacent((5,4),(4,5)) l.isAdjacent((5,4),(5,6)) l.isAdjacent((600,1),(600,2))	True True False True	P
6	isPassable()	l.isPassable(1,1) l.isPassable(4,1) l.isPassable(600,10)	True False row:600 col:10 False	P
7	getAdjacents()	l.getAdjacents((64,50))	[(64, 49), (65, 49), (65, 50), (64, 51), (65, 51)]	P

System – incorporates GUI (view.py)

Strategy: test GUI in context of running application

Test #	Test	Actions	Expected output	Pass /Fail
prep	At command line	python p4.py tests/test7.cfg	After search warnings, GUI opens with blank canvas and statusbar shows "Search unavailable"	P
1	Help Menu	Select Help-About p4 Select Help – Help	Dialog box opens – click OK to close Dialog box opens – click OK to close	P
2	File Menu	Select File – Open Map Navigate to and select alternative mapfile	File open dialog box opens Status bar shows map loading Mapfile opens into GUI – Status shows search unavailable	P
3	File Menu – Quit	Select File-Quit (Rerun GUI with default map: python p4.py)	GUI closes GUI reopens with "OK" status	P
4	Toolbar – Search	Click Search	Path is displayed. Status bar R shows: 'searching' Status bar L: coords, cost, steps and time remaining.	P
5	Toolbar	Click Pause Click Step repeatedly Click Search	Status bar L freezes Status bar R shows 'Paused…' Status bar L updates step by step Search resumes as for Test 4.	P
6	Toolbar	Click Pause, then click Stop Click Reset	Status bar R shows "Stopped." Status bar L shows totals. Yellow path disappears, start and stop remain	P
7.	Keypress	Press 's' Press 'S; Restart from button Pause from keyboard Restart from keyboard Pause from button Alternative keypress (e.g. f, 5)	Path reappears exactly as for 4 Path pauses, exactly as at 5 restarts pauses restarts pauses [no response]	P

PRESENTATION SLIDES

The Brief

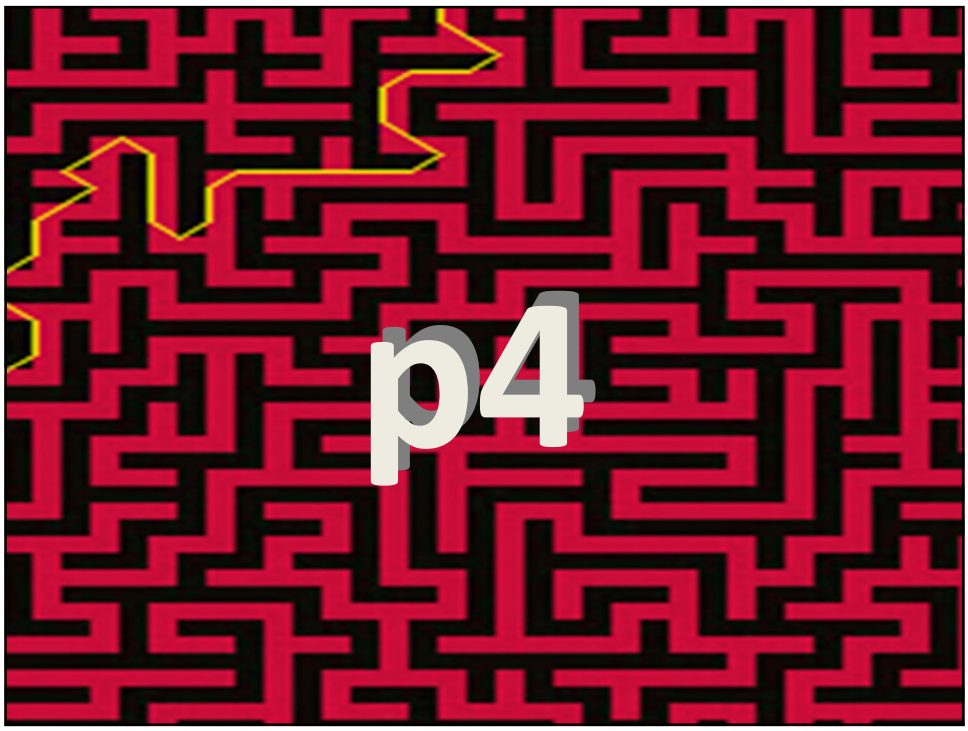
To develop a path planning simulator in which it is possible to:

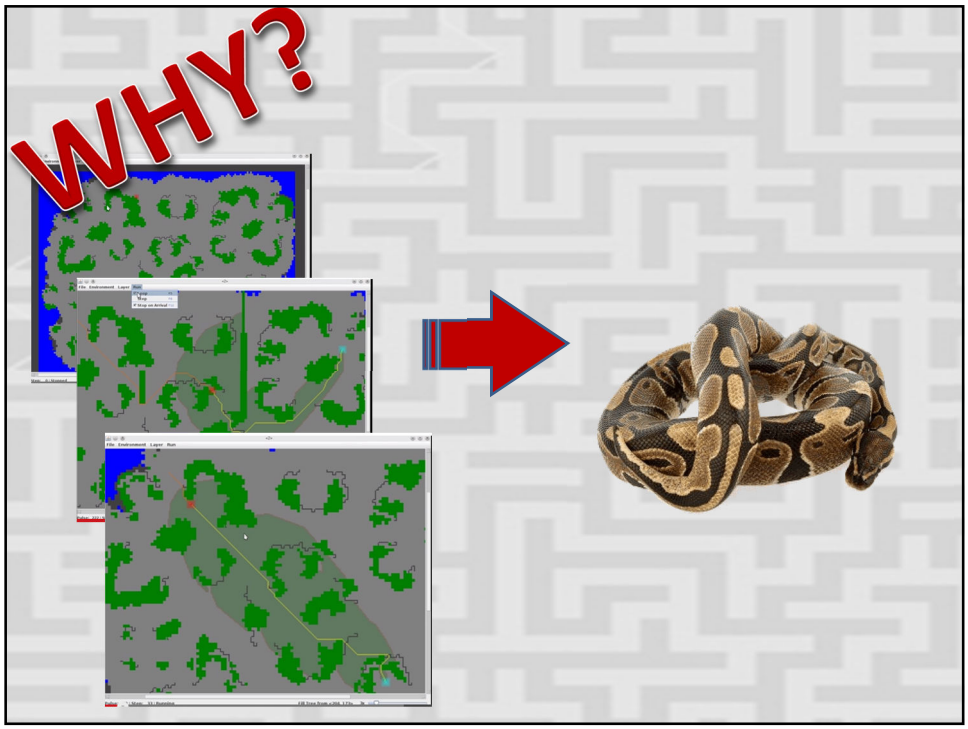
- load different maps in “gridworld” format
- run various search algorithms, and
- display their execution in a graphical environment.

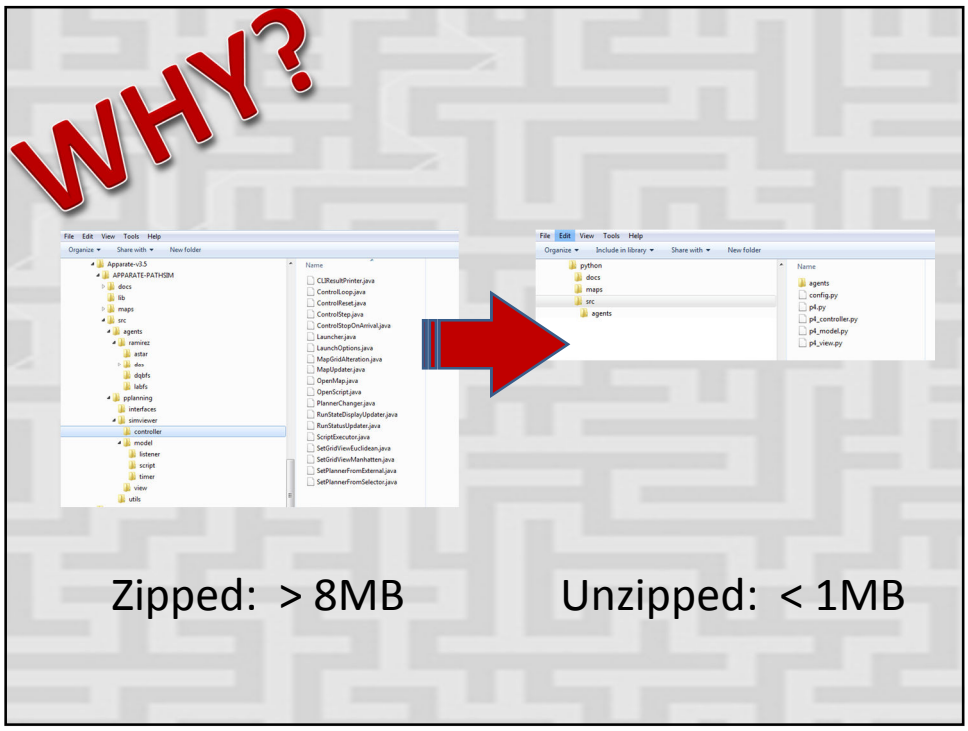
The end result will be similar to Apparate

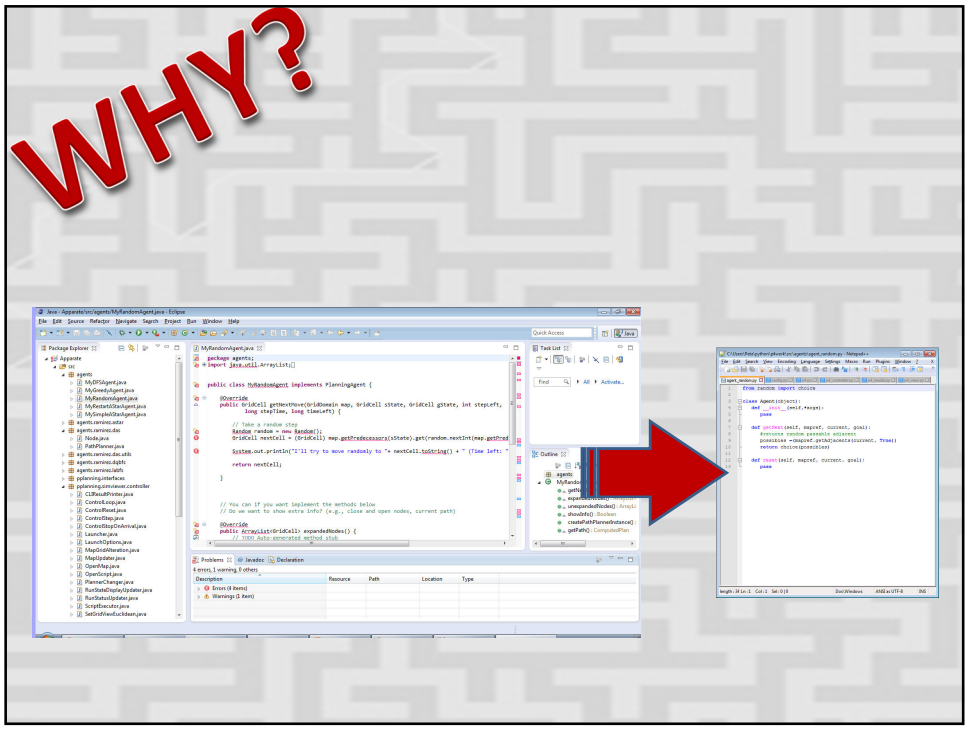
- but based on Python instead.

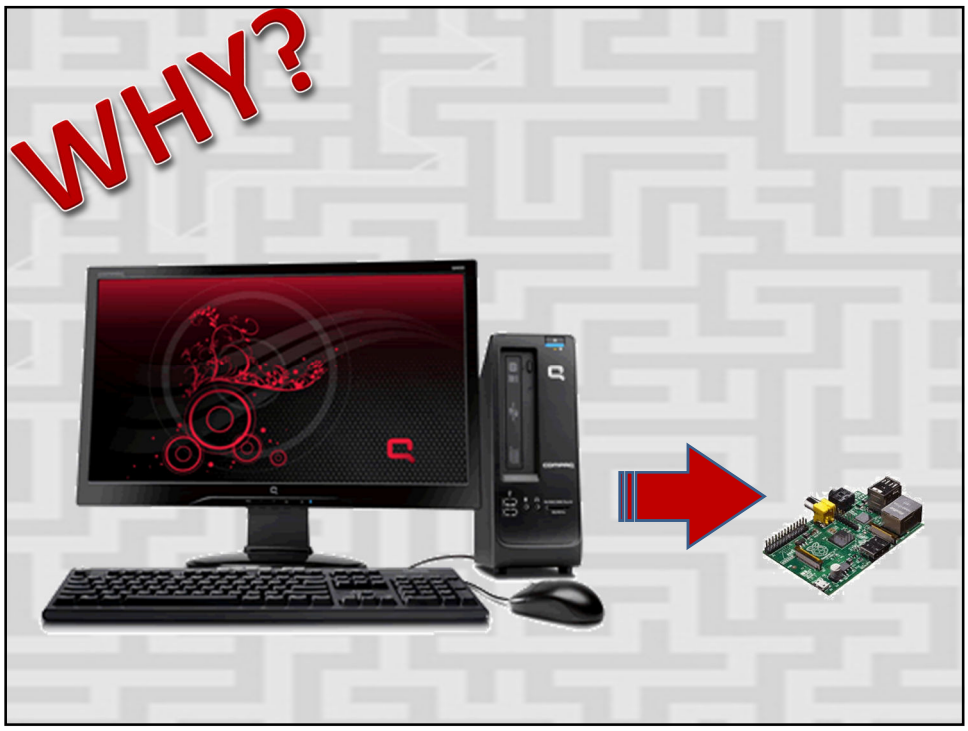
Python Path Planning Project

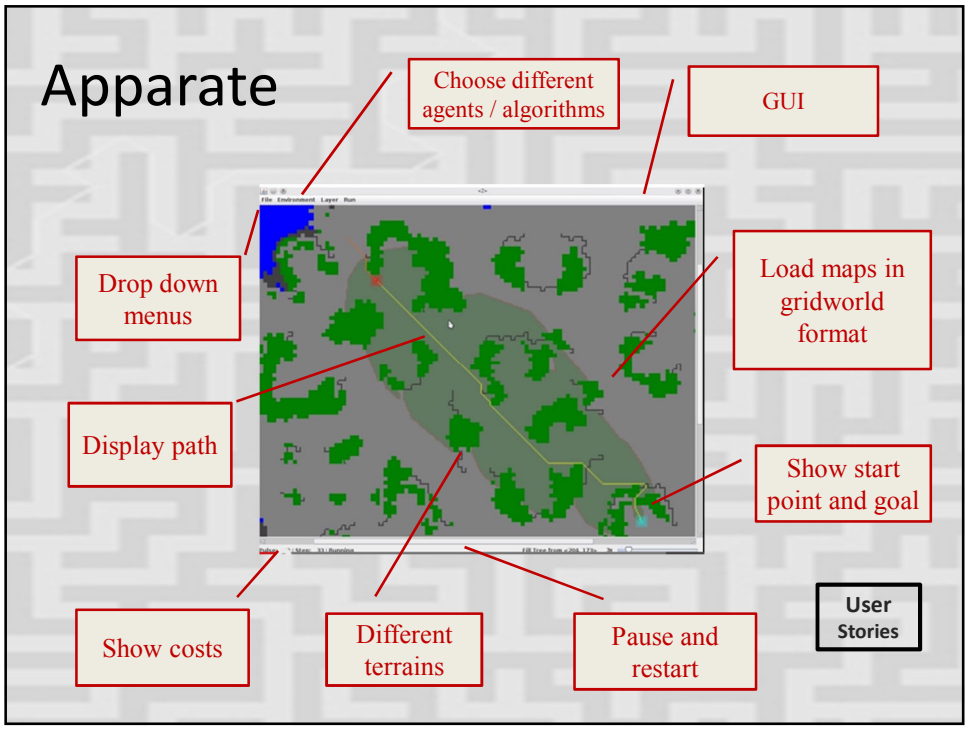

p4


WHY?

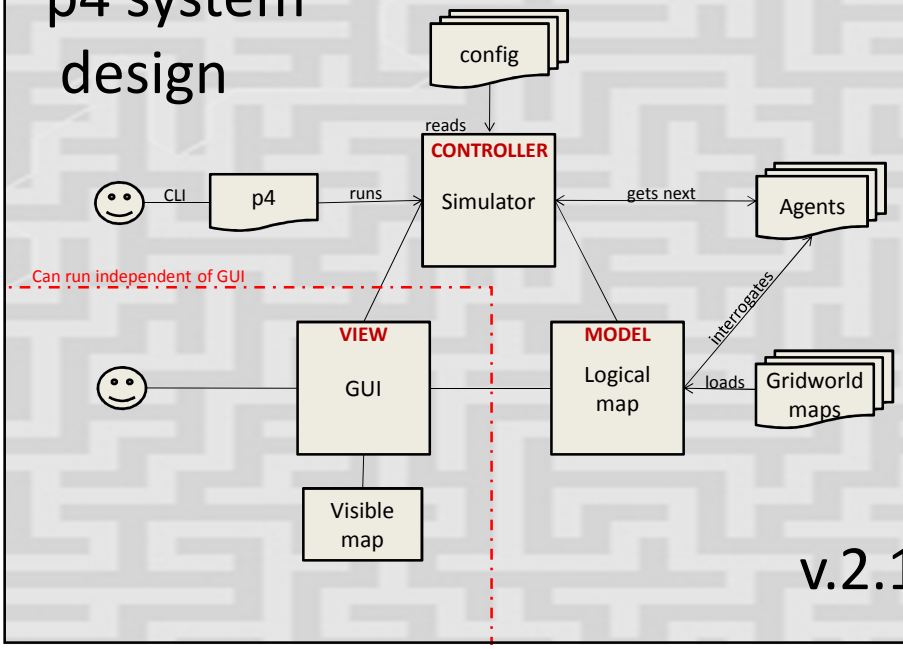

WHY?
Zipped: > 8MB
Unzipped: < 1MB


WHY?


WHY?


Apparate
Choose different agents / algorithms
GUI
Drop down menus
Load maps in gridworld format
Display path
Show start point and goal
Show costs
Different terrains
Pause and restart
User Stories

p4 system design

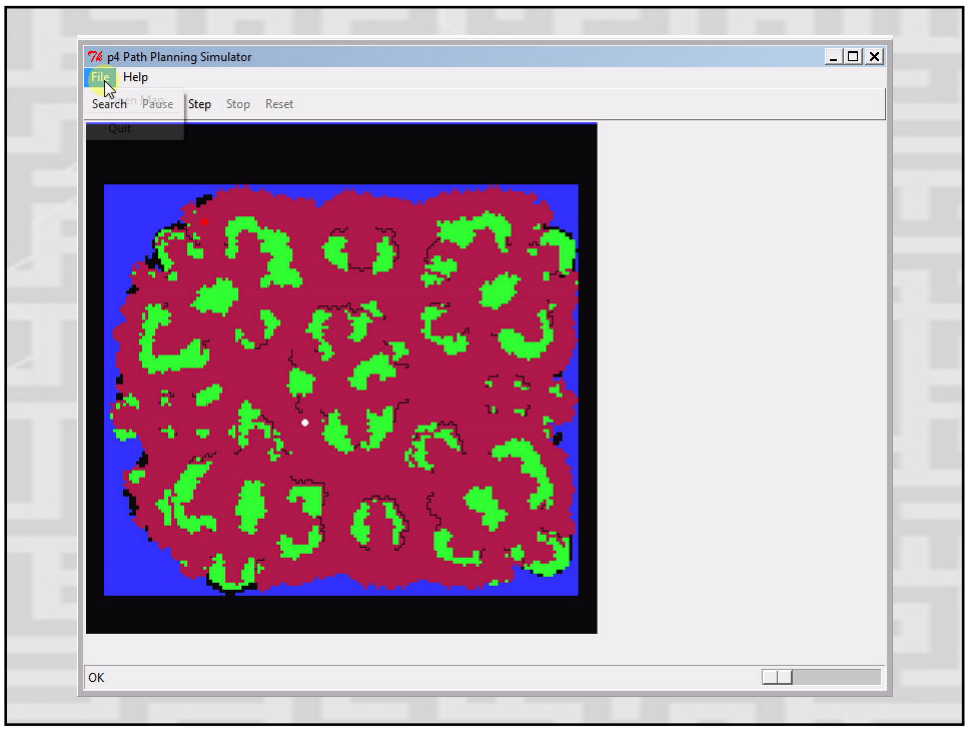


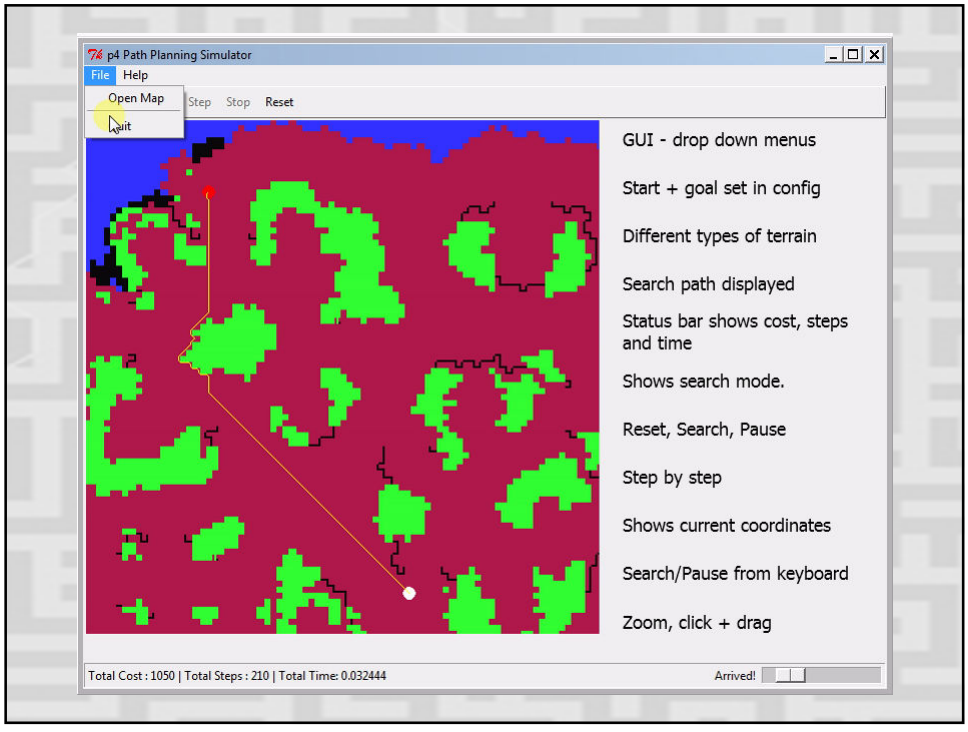
v.2.1

Om

Focus & Discipline!

- Formal requirements
- Schedule
- System design
- Proof of concept
- Test scripts
- User Guide
- Developers' Guide
- Project log


p4 Path Planning Simulator
File
Help
Search
Pause
Step
Stop
Reset
Quit
OK


p4 Path Planning Simulator
File
Help
Open Map
Quit
Step
Stop
Reset
GUI - drop down menus
Start + goal set in config
Different types of terrain
Search path displayed
Status bar shows cost, steps and time
Shows search mode.
Reset, Search, Pause
Step by step
Shows current coordinates
Search/Pause from keyboard
Zoom, click + drag
Total Cost : 1050 | Total Steps : 210 | Total Time: 0.032444
Arrived!

Gridworld Map

```
type octile
height 9
width 75
map
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWGGGGGGGGGGGTTTTTTTGGGGGGGGSSSWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWGGGGGGGGGTTTTTTTGGGGGGGGSSWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGGSSWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
```

END

Config file

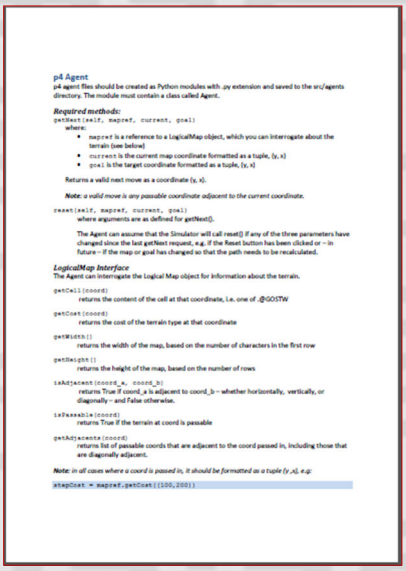
User wants to be able to plug in different agents, different maps and see what happens – so the goal is to make it easy for them to do that.

agent_random.py | config.py | p4.py | p4_controller.py | p4_model.py | p4_view.py | test2.cfg

```
1  AGENT_FILE = "agent_astar.py" #agent file name
2
3  MAP_FILE = "default.map"      #map file name
4  START = (101,120)             #coordinates of start location
5  GOAL = (301,161)              #coordinates of goal location
6
7  GUI = True                    #True = show GUI, False = run on command line
8  SPEED = 0.0                   #delay between moves in seconds
9  DEADLINE = 2                  #number of seconds to reach goal
10
```

END

Roll-your-own Agent



p4 Agent

p4 agent files should be created as Python modules with .py extension and saved to the src/agents directory. The module must contain a class called Agent.

Required methods:

`getNext(self, mapref, current, goal)`
where:

- `mapref` is a reference to a LogicalMap object, which you can interrogate about the terrain (see below)
- `current` is the current map coordinate formatted as a tuple, (y, x)
- `goal` is the target coordinate formatted as a tuple, (y, x)

Returns a valid next move as a coordinate (y, x).

Note: *a valid move is any passable coordinate adjacent to the current coordinate.*

`reset(self, mapref, current, goal)`
where arguments are as defined for getNext().

The Agent can assume that the Simulator will call reset() if any of the three parameters have changed since the last getNext request, e.g. if the Reset button has been clicked or – in future – if the map or goal has changed so that the path needs to be recalculated.

LogicalMap Interface

The Agent can interrogate the Logical Map object for information about the terrain.

`getCell(coord)`
returns the content of the cell at that coordinate, i.e. 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 the 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 – whether horizontally, vertically, or diagonally – and False otherwise.

`isPassable(coord)`
returns True if the terrain at coord is passable

`getAdjacents(coord)`
returns list of passable coords that are adjacent to the coord passed in, including those that are diagonally adjacent.

Note: *in all cases where a coord is passed in, it should be formatted as a tuple (y, x), e.g:*

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

END

Search Button

I thought adding a pause/restart button to the search would be easy – at worst some kind of multi-threading problem.

It took me 2 weeks to come up with these 12 lines of code!

```
def searchStart(self):
    self.btnSearch.config(s
    self.btnPause.config(st
    self.btnStep.config(sta
    self.btnStop.config(sta
    self.btnReset.config(st

    self.setSearchState("Searching...")
    self.searchToggle = True

    #Nested generator returns control to GUI between steps
    def step():
        while True:
            self.simulator.hdlStep()
            if not self.simulator.areWeThereYet():
                self.searchjob = self.after(1, step().next)
            else:
                self.arrived()
            yield

    if self.simulator.areWeThereYet():
        self.arrived()
    else:
        self.searchjob = self.after(1, step().next)
```

END

The Slider

Turns out the 'after' trick was a gift that keeps on giving…

```
def slider(self, event):
    """Internal. Called whenever zoom slider moves. Waits 0.1 sec
    for user action to complete before actioning."""
    #nested method does the actual zooming
    def actionzoom():
        value = self.zoombar.get()
        self.vmap.zoomMap(value,0,0)

    if self.job:
        self.after_cancel(self.job)
    self.job = self.after(100, actionzoom)
```

END

Briefly

p4 is a path planning simulator in which it is possible to:

- load different maps in "gridworld" format
- run various search algorithms, and
- display their execution in a graphical environment.

The end result is similar to Apparate

- but it's based on Python instead.

Any questions?

PROJECT LOG

Week 13 - ending 27/10

Presentation this week and delivery of first draft project documentation - so my focus has been on those two things. Testing is helping me complete the documentation and vice versa - and preparation for the presentation is helping with both too - because it reminds me exactly what I'm supposed to have achieved!

Week 12 - ending 20/10

I have not completed the zoom to my own satisfaction but am keeping to schedule, conducting a code clean up and testing this week. I also have to plan the presentation and am trying to find out exactly what facilities will be available in the presentation room.

Worked on documentation:

- wrote most of the user guide
- some of the dev guide
- appendices for the dev guide - map format and the 'live'' requirements doc - which shows exactly what's done, what's partially done and what's not done at all. The only complete 'miss' is the non-functional requirement sebastian introduced late that the app should really be able to display maps up to 10,000 x 10,000. With the current format there's no way. Even the delay for 1000 x 1000 is horrible.
- wrote some - but still not all - of the test scripts

I had thought the zoom was a complete failiure - but in the course of taking screenshots for the user guide I zoomed the map on a completed search and the path held its position on the canvas really well. It still isn't working of course - but that does mean that the zoom will be possible (a sort of 'if it bleeds, we can kill it' moment!

Met with Sebastian on Friday. We reviewed progress I've made against the requirements and he seemed satisfied with what's been achieved in the time. We talked through the documentation requirements and the presentation. 2 specific 'todos':

- Add a features list to start of the user guide
- Increase 'resolution' of timer for when used on linux (the count down was only about 0.3 of a second - so the time was barely tracked - compared with over about 2 on Windows because the clock timer works differently for each.

Week 11 - ending 13/10

Sebastian is still overseas this week.

I completed the map zoom to the point where I was able to upload it for Sebastian to see as part of my progress report. I can zoom the map in and out and I can zoom the path if it's already on the map. I can also drag the map under the cursor, including changing the cursor to a cross-hatch while dragging. But you can't search while the map is zoomed. Also, you can drag the map out of the 'window' so I need to revisit the logic - when the map is only 1x it should be repositioned exactly so that it fits - at the moment that's not happening. I need to monitor exactly what the bounding box is at all times - at the moment I'm still really trying things out in a proof of concept sort of way and not doing the math!

Having failed to complete the zoom, I compensated by adding the timer and deadline functionality - if a deadline is set in the config file, a countdown timer appears in the status bar. And when the search ends, the total time taken is displayed.

It is only now the completion of the zoom that is outstanding - but it will need a very concerted effort to get it done. I am inclined to start my code cleanup and documentation alongside this. Everything is modular - but I want to make sure the testing is also modular - that is, I want to be able to test the GUI, the map, etc all individually - and that will also help me as I continue to experiment with the zoom. In fact, I would also like to see if I can manage a numpy implementation of the logical map. It should be fairly simple - and would be a good test to see that what I have built really can be modified module by module.

Week 10 - ending 6/10

Coded an A* agent - debugged it and pushed it to Bitbucket. The agent creates the path then - again using a generator - returns one step at a time. I added a reset function to the agent interface to accommodate the reset button and also in case the goal changes or the mapref needs to be reset.

Sebastian is still overseas. I sent a progress report to him, as follows:

> I pushed an update to git yesterday which includes an A* agent - and made some changes to the interface between the agent and simulator to accommodate that.
>
> I've made some progress with the zoom - but at the moment the map is zooming independently of the search path which is not particularly useful! There are quite a few 'challenges' with this element - as predicted.
>
> I still have one more week in which to attempt to resolve the various issues, then I plan a 'code freeze' in week 12 - wherever I'm up to - in order to focus fully on testing - and on preparing for the presentation, which is to occur during study week (week 13 - week-ending 27/10)
>
> I have notes for the user and developer guides but don't expect to finalise them until I prepare the final submission - which is not actually due until the second week of exams (week-ending 10/11).

I have realised that, in spite of working alone, I still need comprehensive test scripts and a clearly articulated coding standard - to make sure my deliverable is truly ready for handover to another developer. The schedule I set out to Sebastian above remains completely in line with the one I originally wrote into the spec. The only addition is the user and developer documentation, which I didn't allow for - but this should be easily accommodated between the presentation and final delivery.

Week 9 - update!

Found a way! And it's not an issue with multi-threading or locks or any of the usual suspects. There are 2 aspects to the solution - one design/theoretical, the other practical:

DESIGN: I'm using the Controller-Model-View pattern and so I wanted the controller to be in control! But - in Python, or Tkinter anyway - once the controller hands over to the GUI, it really has to give the GUI full control of its event management. The controller needs to become not so much a 'controller' as a 'supervisor'. The GUI does everything and just kicks upstairs for the decisions (i.e. business logic).

PRACTICAL: The solution involves using a nested generator which runs a while True continual loop but which yields control to the GUI at every iteration. The outer function contains the next command and that function is called repeatedly, using the Tkinter 'after' command, which you can use to schedule events. When the generator yields, it yields control back to the GUI, which then has the opportunity to cancel the scheduled event if a different button gets pressed!

I wish I could say I thought of all that myself! - but I actually adapted the solution from something I found here:

Sebastian is still away this week. Once I achieved the above (yesterday) I pushed to git and emailed him. This was really part of the 'core' functionality (i.e. to include a stop button for the search). It now works very tidily, and includes pause, continue, stop, 1-step, and reset. But it is a week behind my schedule. Sebastian shifted forward to the 'must-do's an implementation of A* search - if I could achieve that this week I would be back on track - because I need the remaining 2 weeks (before testing) to work on the zoom.

It doesn't now look likely that I will have the opportunity to research further into A* (but I have just today applied to do an Honours year - so if I'm lucky, maybe this is something I can carry forward!)

Week 9 - ending 29/9

I had no idea the multi-threading issue would be so complicated. On the face of it, there seemed very little to do - I had the application running and reporting and basically just wanted to add a 'stop' button. But in fact, turning it from being single-threaded to multi-threaded is a design issue and I'm wondering now whether - to make it really work well - I might not have to go back to the drawing board.

I emailed an update to Sebastian today (24/9) including the following:

I haven't 'pushed' - not because I'm not working at it, but because the code is still broken! Turning it from running on a single thread to being multi-threaded is more complicated than I'd expected - which is all very interesting for me, but not particularly useful for you!!

My sticking point at the moment is the status bar update. On a single thread, I'm able to show all the info we require, but I haven't managed to make the mechanism work for multiple threads - so there's a version that can stop and start and step but without you knowing what the hell it's doing or a version that tells you exactly what its doing but can't be stopped.

What I'm trying to do to fix this is sensible, I think<!!> - which is to send all the status bar updates to be handled by one queue - which runs independently in its own thread. Sensible - except that it's not working.

If I get to the end of this week and still don't have it right, I will try to progress the other issues (e.g. implement the A* search and the timer/deadline, and/or work on the zoom). I will also try to isolate the two 'versions' as actual versions - so that it's possible for you to see and compare them. I don't know if it's possible to put 'branches' on git for that purpose - or even whether that's a good idea? But perhaps wait until end of week to decide!

Week 8 - ending 22/9

Worked through 'must-do' list (item 2 on 130912_p4_meeting.pdf)

Stop/search implementation is not as straightforward as I had thought. There are two python features that I need to juggle, which I've not previously been familiar with - generators and threads.

Generators should be useful for the stop/start/one-step-at-a-time requirements around stopping a running search. They are just kind of diy iterators, which means they maintain their own state so you can just ask for one 'step' at a time and the generator keeps giving you the next one. If you pause - that's ok, the generator remembers where you're up to in case you want to carry on again.

Threads are necessary because while the search is running, the gui needs to be responsive - in order to accept the instruction to stop searching, for example.

On the plus side, it appears that the Python time.clock() function measures CPU time used by the current process - which is exactly the measure Sebastian requires for implementing a 'deadline' on agents returning their path (as opposed to time.time() which returns seconds passed since 1/1/1900 or whatever point in the past. time.clock() doesn't work for Windows - instead returns seconds since the function was called (it all depends on the underlying C call apparently). I'm yet to check and makes sure it works for Linux but hopefully Linux will be Unixy enough. Sebastian had thought that provided the agent activity was isolated into a thread and the thread was timed, that would be OK - but Python threads don't work that way. There's a global interpreter lock (GIL) responsible for switching between threads - they're good for asynchronous behaviour but they typically all run on the same processor so they don't save any 'real' time.

[See http://stackoverflow.com/questions/85451/python-time-clock-vs-time-time-accuracy]

No meeting with Seb this week as he's overseas.

Pushed new work to bitbucket and emailed a progress report (similar to the above).

Week 7 - ending 15/9

Worked to complete implementation of all 'must-do's. In my original task list, I forgot about the mid-semester break and had noted that Sebastian would be away for weeks 9 - 12 - so scheduled delivery of all core functionality for week 8. In fact - because of the mid-semester break - he is away from week 8-11, so week 8 delivery would be too late. This meant bringing forward key tasks planned for next work - specifically:

- delivery of core functionality
- review/re-prioritise remaining stories

I managed to deliver core functionality (and all successfully pushed to BitBucket) with the exception of a search 'stop' option.

At meeting 12/9, Seb reviewed the deliverable and we discussed how to prioritise tasks over the next month. My comprehensive confirmation of our discussion points - as emailed after the meeting - is attached.

130912_p4_meeting.pdf

(I note that Sebastian reworded point (3) to say that I should mark the currently planned path. Then, if possible, also draw the closed and open lists.)

I had not previously noted the specific documentation requirements.

We also briefly discussed the form that the final presentation would take and Sebastian recommended that I create a video in advance - which obviates potential problems with technology issues and nerves on the day.

I note that my meta-A* research has taken a back seat to development tasks. However, in re-prioritising, Sebastian has asked me to complete an A* planning agent (in addition to the 'random' agent) as a sort of 'middle way'.

Week 6 - ending 1/9

1. Sent zipped proof of concept through to Sebastian to save to BitBucket as agreed. In my first pass, I didn't include any maps, so it didn't run - fixed it to present as a working initial distribution by adding a maps directory:

- p4
- code
- p4_poc_v2.py
- docs
- P4 system design v2.0.jpg
- Project Plan_v3.pdf
- maps
- default.map

2. Sebastian had asked me to extend the prototype (proof of concept) to show a working implementation - i.e. so he could see one working run - but when I went to work on that it seemed better to start working on the 'real' project - i.e. implementing what I had designed rather than adding to a proof of concept - that had already been proved! I started working piece by piece arriving at a demonstrable LogicalMap component - which could read in any map and create the map matrix and an info 'dictionary' (with map size and terrain costs), and which could allow the map to be interrogated and modified.

3. At meeting 29/8:

- Sebastian helped me to complete my connection to the BitBucket depository - see notes.
- We discussed the LogicalMap - and it emerged that the data structure I'm creating may not allow for the intense access requirements. The maps I've seen are only 512 x 512, but Sebastian said we should allow for maps of 10,000 x 10,000! I have implemented a simple Python List of Lists (we have to be able to modify the contents of the array, not just populate it and read it). Sebastian suggested using numpy - a Python module dedicated to scientific and mathematical modelling and which specifically caters for fast access to multi-dimensional arrays.
- It was decided that I should proceed with the current structure - and see how it performs - again ensuring that modular design/interface allows for an alternative structure to replace the current one.
- I explained my idea for a multi-threaded A* - I have done some research, but have only found papers from the 90s which (at a preliminary read) suggest using it where there are multiple goals and using a separate thread to work back from each of them. My idea is a bit different - only splitting into multiple threads as the frontier size increases. Sebastian suggested that this might work best if applied, not to a generic graph search, but if applied

to a particular domain (like pathfinding) because then it would be possible to determine mathematically (by comparison of gradients) that one section of the frontier was targeting a different goal from another - and perform the split on that basis.

4. Although not intending to use it immediately, I downloaded and installed numpy and matplotlib (an accompanying maths modelling package), and also pygtk - one of the alternative GUI packages, (a) so I could see if there were any problems doing the install (with Windows there were none) and (b) so they're available for me to switch in for testing/comparison as soon as I have the opportunity.

Week 5 - ending 25/8

1. Extensive research to manage non-functional requirement that the system must not only run on Linux but from the Linux command line. Unable to identify a definite contender, contacted Chris Hoobin (my lecturer for scripting last year - who taught me Python). He confirmed that I would not be able to create a Python GUI that does not depend on X-windows.

2. Developed system design document, including use case. I haven't used formal UML design - the intention is rather to produce a document that will enable me to discuss the design with Sebastian in a meaningful way - i.e. so that it can be agreed and signed off.

3 At meeting, 21/8:

Sebastian confirmed that it was always his expectation that X-server would be running on his Linux box whenever P4 runs - but restated the requirement that the program must run from the Linux terminal's command line.

It became apparent that one requirement has been neglected (although the system design is modular and easily accommodates it) - that it should be possible to run the simulator without any GUI at all. In effect - although the GUI remains a deliverable for this project - it is not really part of the system's core functionality.

We reviewed the system design quite carefully and I stepped him through the base use case.

Amend some of the terminology - to conform with the way Sebastian (and probably all those familiar with the domain) is accustomed to referring to the various components, e.g. what I have called the planner becomes the agent or controller, what I have called the controller becomes the simulator.

The 'logical map' which I had included on the diagram almost as an artefact is in fact a key component and will require the most elaborate interface. This is the piece that the agent-controller will interrogate (albeit through the P4 simluator).

We want to make sure the code and documentation has ongoing existence after this course. Also, he doesn't have access to this wiki so only sees documentation if and when I send it to him or show it to him during meetings. Therefore:

TODO - I am to package up existing documentation and the preliminary 'proof of concept' script as a zipfile and send to Sebastian with my BitBucket login details. He will set up the depository in the first instance. Obviously, this has the additional benefit of providing version control both for the code and the documentation - something it's easy to neglect when you're working on a single-person project.

TODO - develop the prototype to run a simulation (at present it prototypes the interface components only).

4. I'm on schedule in terms of the project plan and task list - the main deliverable this week was the system design. It now needs updating in line with Sebastian's feedback.

Week 4 -ending 18/8

1. Meeting with Sebastian was moved forward to Wednesday - but will remain Thursdays at 2. At meeting:

- Reviewed the project plan
- Amended requirements to make sure they accurately reflect Sebastian's expectations (some items I'd assumed from the Java implementation to be critical turn out not to be - in particular those requirements relating to modifying the map.
- Set priorities using MSCW (Must, Should, Could, Won't) method
- Demonstrated the demo interface

Key issues (already noted but particularly emphasised at this meeting):

- MUST be extensible
- MUST run on Linux
- MUST be able to set parameters for the interface from the command line - e.d. speed and zoom
- MUST be able to zoom in on agent following path - if can't be achieved interactively, need some solution that allows user to see path being taken.

Clarified requirements in relation to interface with planner 'black box' - which is actually a core design issue for this project.

TODO: Test ability to zoom canvas. If has to be done 'manually, how quickly can map be redrawn? does it flicker?

2. Experimenting with Tkinter in Linux, I have found that I have to run X windows first, then run python from the X terminal. I have been searching and searching for workarounds to this - as I'm not sure it will be acceptable. On the other hand, not sure what my alternatives are. Need to clarify the issue with Sebastian as much emphasis was placed (a) on Linux and (b) on running the program from the command line.

3. Updated the project plan in light of our meeting.

4. Separated the task list into its own wiki page, to use for tracking.

Week 3 - ending 11/8

1. Wrote up project plan - inc requirements and timeline and design considerations which point to use of Tkinter for GUI programming if possible - not yet signed off

2. Python GUI research. Included:

- Watched online tutorials
- Worked through tutorials
- Read manual pages

3. Built a simple demo interface using Tkinter and Python 2.7 - with drop-down menus, status bar with zoom slider, and 'map' area.

To be resolved: whether I can zoom the map - and even populate the map - just using native Python 2.7 functionality. I have been trying to avoid extensions in order to maximise accessibility for users and developers - but once non-functional requirements have been finalised it may turn out not to be necessary.

4. Had an idea for Meta A* - derived from observation of Appartite. Visual representation of expanding frontier was reminded me of cell growth. Cells can only support limited surface area - that's why or partly why they divide. A* has a similar problem - once frontier gets too big there are memory issues and too many comparisons. So what if Meta A* - rather than monitoring the performance of the heuristic, simply monitors the size of the frontier. Once it exceeds some predetermined no. it divides into two concurrent searches. And so on, and so on. Asynchronous A*. They share the goal and they share the explored list. But their frontiers expand independently. Whichever one reports back first, wins.

Week 2 - Project allocation and preliminary meeting with Sebastian.

Week 1 - Course briefing.

PROJECT PLAN

P4

Project Plan

Version 1.0

Author: Peta Masters, s3243186
Supervisor: Sebastian Sardina

Contents

Python Path Planning Project

The purpose of the Python Path Planning Project (P^4) is to develop a path planning simulator based on the *Apparate* Path Planning Simulator (which is implemented in Java).

Like Apparate, P^4 aims to support path-planning research by providing an experimental environment, compatible with current research standards, in which search algorithms for grid-based path-finding can be conveniently trialled and compared.

Time permitting, one of the search algorithms available to P^4 will be an implementation of A* involving use of a meta-heuristic TBD based on research undertaken during the course of the project.

Requirements

(MoSCoW: M=Must, S =Should, C=Could, W=Won't)

#	Requirement	Priority
1	Read in any available map supplied in 'gridworld domain' format	M
2	Read in any available map supplied in Apparate format (i.e. including costs for semi-navigable cells such as water, swamp, etc.)	M
3	Display default blank map	M
4	Display map, distinguishing between different features of the mapped terrain	M
5	Optionally, display 'traversable' map, distinguishing only between navigable/ unnavigable areas	C
6	Select between and load any available map from GUI	C
7	Add water to current map	C
8	Add swamp to current map	C
9	Add trees to current map	C
10	Add ground to current map	C
11	Add out-of-bounds to current map	C
12	Add start-point to current map	C
13	Add goal to current map	C
14	Resize map (i.e. resize by dragging interface)	W
15	Zoom in	S
16	Zoom out	S
17	Zoom to selected area (i.e. so that selected area fits bounding box))	C
18	Move map within bounding box window	S
19	Build map, based on script	W
20	Reset to display most recently loaded map	S
21	Run a default search (which may be random or A*) by passing map, current position and goal to planner – planner returns new grid position (n.b. requires simulator/planner interface design)	M
22	Run a meta A* search – i.e. using a meta-heuristic TBD	C
23	Select between multiple search algorithms/planners from GUI	C
24	Specify whether the search is allowed to use diagonal movement (Euclidian) or vertical/horizontal (Manhattan) only. n.b. may require interface with rules module to determine ismovelegal(curr,nxt)	C
25	Display search path on map from start point to goal	M
26	'Animate' the movement of a search agent following the path in discrete observable steps	M
27	Optionally, pause for 1 second at each step (before getting next step)	M
28	Optionally, pause display and Next button after each step then wait for user to click.	S
29	Optionally, display path travelled so far	M

#	Requirement	Priority
30	Optionally, draw/fill area searched (i.e. expanded nodes) n.b. requires inclusion of callback function in P4/planner interface.	S/C
31	Optionally, draw frontier of search (i.e. unexpanded nodes) n.b. requires inclusion of callback function in P4/planner interface	S/C
32	Include callback function in P4/planner interface to draw path (typically from current position to goal)	S
33	Optionally, halt the search	M
34	Display status bar, showing the current step and whether it is running, pausing, stopped or in an error state	S
35	Show grid coordinates of current mouse position	C
36	Indicate currently active function/mode, if applicable	C
37	Permit zoom to be modified interactively while the program is running.	S
38	Permit map terrain (requirements #7-13) to be modified interactively while the program is running	C
39	Permit search path animation options (requirements 26-30) to be modified interactively while program is running	W
40	Permit search algorithm/controller to be switched (requirement 23) while program is running	W
41	Permit all above 'live' changes (requirements 35-53 to be actioned via a script.	W
42	Display search path calculation time (i.e. planner time only) on current processor for most recently completed search on status bar	M
43	Display or log number of base programming steps executed for last search	W
44	Display cost of path for most recently completed search on status bar	M
45	Display search paths generated by multiple algorithms overlaid on one another in different colours to facilitate comparison.	W
46	Perform the search without displaying the GUI	M
47	Return statistics about the mat programmatically - e.g. percentage of map covered by water	S
Constraints and Non-Functional Requirements		
NF1	The project is to be implemented within 10 weeks by one developer, working part time.	M
NF2	Program design must beextensible.	M
NF3	The code must be especially clean and well-commented, and documentation must be complete on handover, so that the project can readily be completed/extended by any other Python programmer.	M
NF4	Whilst preferring an attractive interface, accessibility and ease of use are to be favoured over look and feel.	M
NF5	The system should provide feedback to the user (e.g. via a status bar), so they know the effect of any keypresses or mouse movements and are always aware of the current status of the search.	S
NF6	Deliverable must be able to run on Linux	M
NF7	Must be possible to set options (e.g. map to load, planner to use, speed and zoom) from the command line	M
NF8	Must be able to support interrogation of maps up to 10,000 x 10,000 without any observable drop-off in performance	S

Goals

To port the planner from Java in order to exploit the perceived benefits of Python, including:

- an easy and clean environment in which to prototype search algorithms
- numerous existing libraries, especially useful in the context of scientific research
- an expanding user-base, which will help ensure availability of developers to maintain and grow the project.

Scope

While aiming to exploit the particular capabilities of Python, the purpose is primarily to reproduce Apparate's functionality; by being mindful of Apparate's design choices, it should be possible to avoid getting stuck on problems that have already been solved.

Like the Java application, P4 will adopt an MVC design pattern. At each iteration cycle, the model for the map is revised and displayed. The controller then passes the map, agent location and goal to the planner and the planner returns the next move.

Depending on progress, time constraints and requirement prioritisation, the planner's strategy can be treated as a 'black box' which might return random moves or follow a path planned with A*. Ideally, however, some version of the planner will be implemented, capable of demonstrating A* and a meta-A*, research for which will be undertaken as part of this project.

Choice of programming language will be guided by the following considerations:

1. In order to take maximum advantage of existing Python libraries and the growing community of programmers, the project will be implemented in CPython – as opposed to JPython or Jython. If it were to become necessary to reuse Java classes (an advantage of Jython), there are extensions available such as Pyjnius and Py4j – but use of these extensions and, in fact, reuse of existing Java classes is out of scope for this phase of the P4 project.

2. In order to maximise accessibility – both for users and for future developers – Python 2.7 will be used, rather than Python 3.3.

3. It has been noted that Python's native GUI, TKinter, lacks 'smooth' widgets and various extensions have been suggested. Having researched this issue, it appears that the much 'sexier' ttk widgets, which are fully implemented in Python 3.3, are also available to a lesser extent as an enhancement to Tkinter's look and feel from version 2.7 onward and I intend to take advantage of these - proposing to use Python 2.7 with Tkinter and ttk. This decision may of course be revisited if the interface for any reason becomes unworkable.

Roles & Responsibilities

Researcher/Developer:
Peta Masters, 0430 887 382, s3243186@student.rmit.edu.au

Supervisor:
Sebastian Sardina, Room 14.8.7D, sebastian.sardina@rmit.edu.au

Weekly Task List

Week:	3	4	5	6	7	8	9	10	11	12	13
Create project plan	✓										
Research existing implementation	✓	X									
Research available technologies	✓										
Preliminary design		X									
Develop proof of concept		**X**									
Obtain detailed requirements as user stories		X									
System design			**X**								
Build 'must' (prioritised) functionality			X	X	X						
Research meta A*		X	X	X	X	X	X				
Test and deliver core						**X**					
Review/re-prioritise remaining stories						X					
Develop additional stories ('should'/'could')						X	X	X	X		
Integrate stores as they're developed							X	X	X		
Finalise & Implement meta A* algorithm								X	**X**		
Integration and testing										X	
Prepare presentation										X	
Deliver presentation & final submission											**X**

NOTE: In shaded weeks, Sebastian will be available only via email.

Project Management

Although working with a 'team of one' I intend to adopt an agile-esque methodology, insofar as I aim to obtain all requirements in the form of prioritised user stories. I will build a deliverable core application as early as possible and thereafter aim to implement – and integrate – each story or group of stories as discrete functions/modules.

This approach should make it possible to deliver the final presentation and submission at any stage after the core application has been realised. It will also minimise the documentation load – important given the time-constraints. A further advantage of the simple story-based requirement format will be to minimise risk of miscommunication.

Weekly meetings will take place at Sebastian's office – 14.8.7D – each Thursday, 1400-1500 and the Project wiki will be updated regularly with the main points arising from those meetings.

The wiki will also be updated with ideas and links in relation to the research element of this project – i.e. the meta-A* search – and used as a "project diary" to ensure that, as a team of one, I am careful to monitor my own progress.

Risks

Sebastian will be away during the final weeks of the project, immediately prior to delivery. There are two ramifications of this: I won't be able to walk him through the application in order to obtain approvals/story changes; I won't have the benefit of his advice during a crucial phase.

Mitigation: make sure we are able to exchange files (eg. via GitHub) before he goes. Ask Sebastian to deputise – if possible – someone else who has been involved in the Appartite project to give advice on his behalf while he's away.