p4
Developers' Guide

Version 2.0

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 and the README.

Project Overview

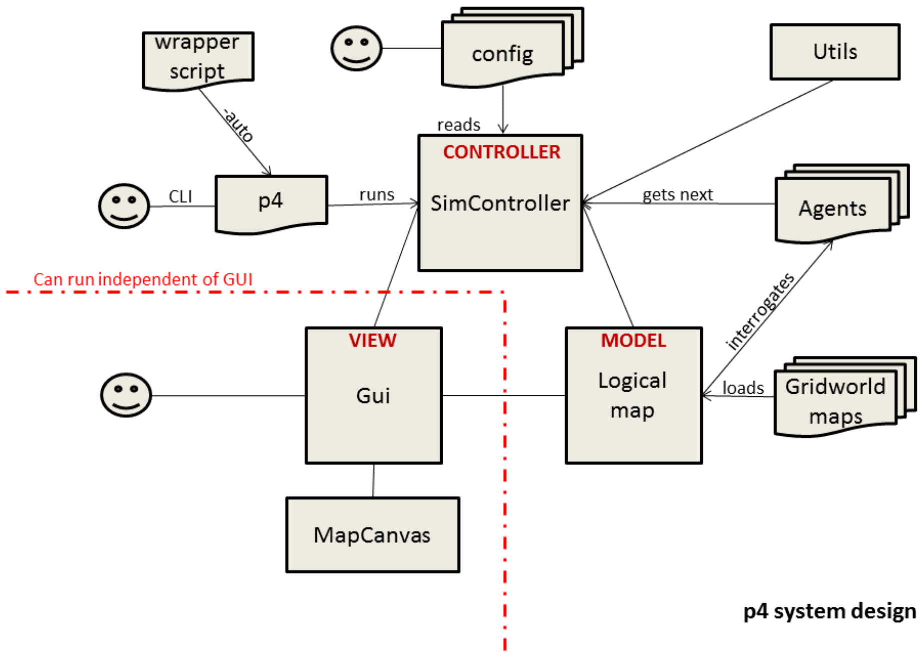
Undertaken as an undergraduate Programming Project for COSC2409 in Semester 2, 2013 and continued into the Summer School 2013/14, the brief was to build "a path planning simulator in which it is possible to load different types of maps, run various search algorithms, and display execution in a graphical 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 scripts, utils, and config files.



p4 system design

***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
SimController	P4 controller. Handles all the business logic, including search. Maintains interface between Agent and Gui.
Agent	Given map reference, current position, goal, and time remaining, 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
Wrapper	The ability to set all options from the CLI, including an –auto switch, allows for automated testing, using a wrapper script. An example (test.py) is included in the build.

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 ideas for future development.

p4.py
Launches P4 by initialising the Simulator and passing in the name of a config file or a list of equivalent parameters.

- Uses argparse to accept the config filename or the individual parameters as command line arguments (see README for full list of switches and their defaults).
- If a map filename has been passed in, assumes there is no config file and passes rest of arguments through to the simulator.
- If no map file is named, checks for a config file. If the config file doesn't exist, 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 – any future options added to the config file can be added here to be passed in via argparse.

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 a cfg 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.cfg 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.

SimController

- Reads in the config file or 'assimilates' the arguments and initialises Agent and LogicalMap
- If AUTO = True, supresses all info messages to the command line, so that a wrapper can be used to automatically process the final output. Auto output is formatted:
  cost; steps; timetaken; timeremaining
- If GUI = False, performs single search and reports result to command line
- If GUI = True, imports view module, initialises GUI, and waits for user action.

SimController 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.

SimController 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 for the GUI.

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 in the format abbreviation: term (i.e "G": "ground", etc.). When the costs are read in from the map header, they are switched with the term that they match so that subsequently, costs can then be retrieved based on the actual content of any given cell (e.g. "G": 1, "S":10, etc).
- Impassable cells are costed as float('inf').
- 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.
- Extra functions have been added to LogicalMap to handle pre-processing for extensions to JPS (see thesis) – they are located in the map model rather than in the agents themselves to ensure consistency – and because they may be useable by other agents down the track.

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.

MapCanvas

Gui creates a mapcontainer frame, which contains the MapCanvas object. The controller always accesses the map via the Gui BUT acts directly on the Gui object when coloring the map to avoid unnecessary timelag.

- The VisibleMap is realised as a Tkinter.Canvas object.
- 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, start and end points, and all additions to the map are also bitmapped and must be tagged by the program if they're to be removed (i.e. redrawn).

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.

Utils

The only real utility currently is the Timeout function. A timeout is supposed to occurs if 2 x the deadline elapses and the agent still hasn't returned a result. Currently, this is only implemented under Unix. Utils holds a dummy class for Windows. WinTimeout()is just a placeholder but without it, we would have to check at every step of the search whether or not we're running on a Unix machine. This way, there's a once-and-for-all check at the top of p4_controller to sniff out the os.

Limitations/Proposed Enhancements

Search

- Simulator does not validate the Agent's move to make sure it's legal. It checks for corner-cutting or an impassable cell and costs the path as 'infinite' if either occur but it doesn't in fact validate that the next step is adjacent to the last! The agents have to be self-policing.

CLI

- The CLI parameters were implemented in a hurry. As noted in the codebase, there's duplication. The way parameters are set needs to be rationalised – especially if they are also to be settable from the GUI, which most of them could and should be.

- There should be parameters to set corner-cutting (from config and GUI) and to choose which timer. Currently, corner-cutting is prohibited – and that's hard-coded (the model costs a cut corner at infinite cost) and the timer type is set as a parameter in p4_utils. [n.b. the timer type was to be configured automatically based on o/s – but timer() must be used with Mac OSX for precision, while clock() is preferred for Linux.]

VisibleMap

- 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).

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

Known Bugs

#	Description	Status
1	Bug or 'feature'? - It's possible to restart a stopped search even though buttons are inactive by pressing the 'S' key.	
2	When you click on the map to set the start point or goal, sometimes the map moves – you can drag it back to where you want it, but it's annoying.	
3	Also when you set a start point or goal, there is an automatic reset – which means, if you were zoomed in to set the goal, say, the map zooms back out again as soon as you've done it.	
4	Occasionally, the play/pause button gets out of synch. It doesn't happen often – it must be some particular combination of button presses – but it needs investigating.	

Coding Standards

To keep my code consistent, I have applied the following variable-naming conventions, etc.:

- dict keys - caps, e.g. MAP_FILE
- modules - lower case, e.g. p4_model
- classes - capitalised, e.g. LogicalMap
- methods (functions) - internally capitalised, e.g. areWeThereYet()
- attributes - lower case, no word divide, e.g.pathcost
- prefix local attributes/methods with _
- handlers (in controller) - always begin 'hdl', e.g. hdlReset
- interface elements (in GUI)- always begin with abbrev for interface element, e.g. btnSearch
- color - always use American spelling

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.
- Stories marked W in previous versions of this document have been removed from the list.
- 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
7-11	Add water, swamp, trees, ground, out-of-bounds to current map	C
12	Add start-point to current map	C
13	Add goal to current map	C
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
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

#	Requirement	Priority
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
42	Display search path calculation time (i.e. planner time only) on current processor for most recently completed search on status bar	M
44	Display cost of path for most recently completed search on status bar	M
46	Perform the search without displaying the GUI	M
	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 Apparate, which is adapted from that used for Nathan Sturtevant's gridworld benchmarks (Sturtevant, monivingai.com).

In common with the gridworld 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 Apparate 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 P4 takes the actual width/height of the grid – i.e. imported content – as the map width/height.*

Example

```
type octile
height 9
width 75
map
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWGGGGGGGGGGTTTTTTTGGGGGGGGSSSWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWGGGGGGGGTTTTTTTGGGGGGGGSSWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWGGGGGGGGGGGGGGGGGGGGSSWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWGGGGGGGGWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
```

References

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