Animating Search Algorithms

By Ian Brown, Jack Histon, Colin Jackson,
Jennifer Jones and John Sanderson

October 14, 2011

1 Project Description

Dr. Katie Atkinson provided the following brief for the project:

> There are a number of well known Search algorithms that are used in Artificial Intelligence applications. The aim of this project is to develop a suite of animated graphs that implement different search algorithms, such as depth-first search, breadth-first search, greedy search, A* search etc. It is envisaged that such a tool could be customisable to different user input and as such, the tool could be used as a teaching aid.

A piece of software will be created to animate a number of search algorithms. The finished software will be a teaching aid, used by both educators and students. The tool will provide the user options of loading preset graphs whilst providing the functionality to edit them or create new custom graphs. It is essential that the interface is intuitive and that the animation makes it clear to the user how the algorithm is running.
From the above extract the following aims are determined:

- Provide a suite of animated graphs - Animations will show on a pre-rendered graph how each search algoroithm will behave. Different algorithms will be able to be run on the same graph, each animating the graph such that it can be understood by students.
- Provide aid for teaching search algorithms - When teaching search algorithms a teacher will be able to use the suite to load premade relevant graphs for the lesson. The students will have a visual aid with which they will be able to download and experiment on.
- Provide tools for customisation - A user will be able to load preset graphs built into the system, but also edit and create new graphs.

2 Statement of Deliverables

2.1 Anticipated Documentation

- Project Specification - High level description of the project, defines what deliverables are to be produced and outlines a plan for the preparation, design and implementation stages.
- Progress Report - Describes the current status of the project and what is yet to be completed.
- Project Report - It encapsulates the entire project, from its specification and background, through the design and realisation, to the final evaluations and learning points.
- Individual Report - Outlines each team member's experience during the project: what they contributed, what they enjoyed and disliked, what was troublesome or not, the perceived synergism of individuals and what they have learnt as a result of this project.

There is an additional presentation for each document deliverable.

2.2 Anticipated Implementation

This section shows a list of features the software should provide in order to meet requirements given. The software needs to attain:

- An ability to show custom and preset graphs to represent a particular search algorithm. The software will be used in a teaching environment. Therefore, this ability will allow search algorithms to be taught with clarity.
- An ability to edit existing graphs. A preset graph will be given to a user on selection of a search algorithm. They will be able to start a blank canvas for custom graphs as well.
- An ability to select different algorithms, switching between them seamlessly. Graph data will be retained so users may return to a search algorithm when switching from one to another.
- An implementation of key[2] search algorithms. For each algorithm providing a stereotypical graph, graph modification tools, and a play/pause button. The following are the required search algorithms to implement:
  - Depth First Search.
  - Breadth first search.
  - Depth Limited search.
  - Iterative Deepening
  - Bi-directional
  - A*
- An ability to animate the progression of a search algorithm. To chart paths toward a solution, the software needs a step by step timeline. This will show different stages of an algorithm, eventually reaching a goal state.
- An ability to display each search algorithm with corresponding metrics. Such metrics are:
  - Completeness - does the search algorithm always return a solution?
  - Time complexity - the number of nodes the search algorithm has generated/expanded.
  - Space complexity - The maximum number of nodes in memory.
  - Optimality - does it always find the optimal solution?

There are also some desirable features. The software will try to attain:

- An ability to save and load custom graphs. This will give a user the ability to save work, coming back to it later.
- A Colour blind mode. Some users of the system may be colour blind. Distinguishing different states of the system may be harder for such users. A change in colour may rectify this.
- A theme modifier. As the anticipated software will be primarily used for a teaching aid, it will have various mediums on which it will be displayed. An ability to change the theme of the software is beneficial as it can be tailored to the specific medium. Such mediums the software may be used on are lecture room projectors and university campus computers.

2.3 Anticipated Evaluation

- Testing - This will cover the implemented search algorithms; including graphs that may prove difficult. It will also ensure that all features and functions of the UI work as expected.
- Specification Comparison - Allows a comparison between the end product and the aims.
- End User Acceptance - Acceptance from the end user is paramount. If the end user does not feel that the product is suitable for teaching then the project has not been entirely successful, even if all requirements have been met. Such end users will be the project supervisor, and module co-ordinator. There will hopefully be a chance to present the software to students to gain feedback and to gauge its usefulness.

3 Conduct of Project and Plan

3.1 Preparation

There will be background research into each of the algorithms we implement, as we will need to fully understand them in order to animate them correctly and informatively. This is trivial with the simple algorithms (such as DFS/BFS), however, the more complex algorithms, Such as A*,[2] will require more preparation for correct animation. There will also be research into the graphics library to use and how to best organise the user interface. The only external data required for this project is graphs to test the algorithms on, these can be arbitrarily created by us to suit the needs at the time.

3.2 Design

We will be employing a Rapid Application Development approach to producing the software. This suits the short timeframe we have available for implementation by minimising the time spent on the traditional design phase by only performing minimal design before actually starting the implementation. As implementation progresses the design documents are created and modified accordingly to suit any changes that have taken place with the direction of the project[5].

3.3 Planned Documentation

- Diagrams. We will compose a selection of diagrams describing the various mechanisms in the final software. These will define classes, data structures, relationships and use cases. Most of these are best presented as UML diagrams[7].
- Pseudo-Code. The algorithms will first be described using a high-level pseudo-code language.
- User interface mock-ups and wireframes. These will show sections of the UI and their anticipated final appearance and simple functions.

3.4 Implementation

- Hardware. The software is intended to be used as a teaching aid, so it should run on as many platforms as possible. The main target platform, however, will be a budget-spec Windows computer. Function through the web browser on tablet devices would also be desirable, though not required.
- Software. The project will be coded in CoffeeScript (A ruby-like language that compiles into JavaScript[4]). CoffeeScript is based on JavaScript[1] and can be used in any modern web browser (i.e. Released in the past few years with SVG/VML support). This means platform independency can be achieved. CoffeeScript also has capable GUI libraries at it's disposal[3]. CoffeeScript is chosen as the implementation language because of it's more conventional object oriented model when compared to vanilla JavaScript's prototype model. It's also succinct and uses a simple, intuitive syntax.
- Testing. Each of the implemented algorithms will be tested alongside a suite of graphs yet to be composed. The graphs will contain a selection of expected difficult scenarios (e.g. cycles, disconnects, etc)[6]. Simple testing will be carried out on any UI fields too (such as out-of-bounds, presence, invalid data, etc).

3.5 Risk Assesment

- One of the major challenges in this project will be implementing a fluid and easy to use interface which demonstrates the algorithms all in an effective and aesthetically pleasing manner.
- Most of the group has never worked with JavaScript or CoffeeScript, so there will be the initial hurdle of understanding.
- There could be a problems with merging of code within the group. This can be slightly avoided by providing a standard interface class for everyone to work up against.

References

[1] Jeremy Ashkenas. CoffeeScript documentation. `http://jashkenas.github.com/coffee-script` [Accessed 1st October 2011], 2011.

[2] Katie Atkinson. Search in AI lecture notes. University of Liverpool, COMP219, 2011.

[3] Dmitry Baranovskiy. Raphaël reference. `http://raphaeljs.com/reference.html` [Accessed 3rd October 2011], 2011.

[4] Alex MacCaw. *The Little Book on CoffeeScript*. 2011. `http://arcturo.github.com/library/coffeescript` [Accessed 5th October 2011].

[5] Martyn A. Ould. *Managing Software Quality and Business Risk*. Wiley, 1999.

[6] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2003.

[7] Perdita Stevens and Rob J. Pooley. *Using UML: software engineering with objects and components*. Addison-Wesley object technology series. Addison-Wesley, 2006.