|  |  |
| --- | --- |
| Name | Sam Endean |
| Student number | 12011876 |
| Project Title | Turn based strategy AI within 3D space |

# Description

The system will be built with the idea of a turn based 3D space battle between two teams in mind, meaning the combat focus will be that of ship to ship combat, with one or both sides being controlled by AI players competing to destroy the other:

* The entire space will be comprised of 3D volumes which will act as the tiles seen in other turn based strategy games, meaning movement will be limited to going between these volumes or rotating with them.
* Tiles will be randomised to have effects such as black holes slowing movement or a comet causing damage to anyone passing through said tile.
* The areas of AI that will be addressed shall be path finding along with tactical combat.
* The AI will generally use the “Goal driven agent” model of AI as opposed to that of a “Finite state machine”.
* The fog of war will limit the initial vision of the fleet, giving both sides the necessity to explore the space from the outset.

# Research and background

There is a large selection of turn based strategies on the market, however this project is an attempt to tackle the problems of turn based strategy AI within a space of three dimensions. As of writing this there are no games of this description (the closest being stationary, gravity based artillery games). Thusly this is exciting new ground.

To date I have researched game AI text books along with academic papers and online sources:

One such source was a text book on expert systems. It introduced several different search algorithms, such as hill climbing, Dijkstra's algorithm and A\*. This source also outlines the STRIPS and MYCIN goal based systems (the latter of which was used by doctors to help prescribe antibiotic medicine). (Jackson, 1999) In terms of usefulness, this book gave a good introduction to the algorithm that I wish to use as the basis for the pathfinding in the system, the A\* algorithm. Not only this, but the STRIPS system has some very useful concepts such as maintaining a list of goals instead of following a single goal to completion as with a finite state machine. However the extent of this source’s usefulness is quickly reached, as the source is fantastic for introducing concepts and ideas to be read into further, yet it lacks substance in the way of practical application and example implementation.  
  
Another source I have researched through is “Programming Game AI by example”. (Buckland, 2005) The work serves as a more specific and focussed display of programming AI for games, especially with its examples of implementing pathfinding algorithms which show explicitly how A\* and other such methods are coded. Furthermore the book has a large section on “Goal driven agents” which, having read, caused the planned method of implementing the individual agents of the system from the finite state machine model to that of the Goal driven agent model, as these are less rigid and much more adaptable to changing circumstances whilst carrying out a list of goals according to priority along with the cost of carrying out such actions. This source is the most useful work examined thus far; it has not only demonstrated practical examples of pathfinding within a tiled plane using the A\* algorithm, but has also provoked much thought and contemplation in terms of how the AI system would mechanically be composed. Such contemplation has causing a very important and positive change in direction when it came to planning the arrangement of logic to make the system function.

# Objectives

### Project objectives

This project will achieve:

* An AI system that can traverse, explore, and comprehend three dimensional space reliably.
* Agents that can prioritise and carry out tactically advantageous goal in terms of ship to ship combat.
* A system which may be utilised by a user in order to battle against an AI opponent, or alternatively may be utilised by another AI allowing a user to watch the battle occur.
* A (pseudo)randomly generated combat environment which will cause skirmishes to unfold in a differing manner each match.
* A tiered AI system whereby the otherwise autonomous agents are given tactical priorities by a strategic command layer that oversees the entire battle.
* Display the three dimensional environment in a sensible, easy to comprehend way so as to reduce disorientation of the user/viewer.

### Research objectives

The research objectives for this project are to find out:

* Implementing/programming pathfinding through the A\* algorithm.
* Implementing/programming the behaviour of goal driven agents.
* The creation of a VERY simple turn based skirmish game in which the AI will operate.
* How to draw/produce the chosen tiling three dimensional shape and how to tile it perfectly.
* A general knowledge of real world naval tactics for the strategic tier to recommend to the agents within the world.
* How games such as the Civilisation series manage their combat AI systems. (Which happen to also be within a tiled environment)

### Learning Objectives

Personally, this project is being carried out in order to learn:

* The details of creating somewhat competent AI for a turn based strategy game.
* The experience of innovation within a beloved genre of games.
* More methods used in order to give the illusion of intelligence within a piece of interactive computer software.
* Time management when working on a large individual project with nearly-complete independence.
* How to prefect methods of research and knowledge acquisition, which will be transferrable to for any future project in life, not limited purely to software development.

# Methods, techniques, tools and processes

This project will be split into several sections when it comes to tackling its implementation:

Firstly will be the creation of the world in which the AI will operate. As this is not a low level graphical project, the aim is to use the Unity engine as a framework in which to toil as it allows for the quick and easy visualisation and communication of the systems elements. Whereas using an API such as DirectX would require a lot of low level development and therefore will detract time, effort, and resources away from the actual focus of the project, which is the high level interaction of agents within the world. Furthermore the initial creation of the world will by drawing regular dodecahedrons in code to produce an outline of said shape. The regular dodecahedron was chosen as it is one of the five platonic solids, it allows for many different options as to what direction an agent may move in, and also forms a clear path in a straight line in any of these directions. Objects will move between the dodecahedrons by simply transitioning through any (un-rendered) face of the current dodecahedron into an adjoining one in any of the twelve possible directions. The way this is to be managed is by assigning an I.D. to each dodecahedron upon its creation, it will also contain the I.D. of each other dodecahedron it borders and upon which face it borders the other dodecahedron with. (The numerical layout of the dodecahedrons faces will follow that of a twelve sided (“D12”) die.) In some ways this can be seen as a vast expansion of a second year project that saw the creation of binary trees and linked lists, alas in this case with many more ‘branches’ from each ‘root node’.

As for displaying the world to the user, being in three dimensional space can prove very confusing and disorientating. Ergo the display will contain a small tri-axis object in the corner in order to show the current orientation of the camera in relation to the world. This will be achieved by applying the current rotation of the camera to the display object. (This is seen in many other pieces of 3D rendering software) Also in terms of displaying the honeycomb itself, the user will have the ability to select and highlight a certain object, all objects in the world that are in a closer layer to the camera than the currently selected object shall not be displayed. The method of doing this shall be through having an invisible plane object that is set to face the camera at the origin of the currently selected dodecahedron. From this object the system will detect all dodecahedrons that intersect with the plane and then will cause all other dodecahedrons within the view frustum between the plane and the camera to not be displayed to the user.

Secondly the project will have the pathfinding feature constructed using the A\* algorithm, the formula for the A\* algorithm is f(n)=g(n)+h(n) where g(n) is the known cost of getting from the initial node to n and h(n) is the heuristic estimation of reaching the goal from n, as long as the heuristic estimation never overestimates the cost of reaching the goal, the system should find the optimal route. Having a background in A-Level maths, including a full module on algorithms and in particular Dijkstra’s algorithm. The fact that the A\* algorithm is a generalisation of Dijkstra’s algorithm (cutting down the size of the subgraph to be explored through use of the heuristic function) means that my skills and pre-existing knowledge is mostly there already.

Thirdly the agent’s goal driven framework will be implemented by having goals that are either atomic goals, and so are a method of completion unto themselves, or composite goals, which may split down into further composite goals or atomic goals. Each goal class works through four uniform functions (and a fifth one for composite goals): First is the *activate* function, this is where goal planning takes place and at first glance appears similar to the *state::enter* method utilised my many finite state machines. However here the *activate* method differs as while the *state::enter* method is only called once (upon entering the state) the goal is able to call it’s *activate* any number of times in order to redo its plans if the situation demands. Secondly the *process* function, called every update step, returns an enumerated value representing the status of the goal: *inactive* means the goal is not the highest priority and is waiting to be activated. *Active* shows that the goal has been activated and will be processed within each update step. *Completed* signifies that the goal has been completed and will be removed during the next update. And *Failed* shows that the goal has failed and must either redo its planning or be removed during the next update.

The way that these goals will work shall be through having a priority queue of current goals, with survival goals having a higher weighting than an explore goal.

Having worked with finite state machines in the past I nearly have the skills/knowledge required to implement this feature as it is the next evolution from the other AI work I have been used to such as the group task of the level two module Game Engine Architecture, in which I was the person in charge of the engine’s AI features and implementing the finite state machines for the system.

In order for the system to be created thoroughly, one part of the AI shall be worked on at a time, for the prototype due in February the main focus will be on completely finishing the A\* pathfinding, along with presenting the attacking mechanics. This means that instead of overwhelming the project with adding many features at once, the process may be iterative and allow for focus to be solely in one area.

# Risks and issues

|  |  |  |
| --- | --- | --- |
| Risk | Mitigation | Contingency |
| Failure to implement the A\* pathfinding algorithm. | Research throughout the project until the knowledge is solidified and the feature is implemented. | Dijkstra’s algorithm will be a fall back that I already know with confidence and would simply require more computing power than A\* (Which is forgivable within a turn based system but is definitely not ideal.) |
| Too long spent researching and implementing the tiled 3D space. (through drawing dodecahedrons in code) | As a guideline for the shape, a 3D model of a regular dodecahedron along with a physical D12 die have been procured. These should assist with visualisation and comprehension. | A simpler shape such as cubes could be a suitable backup plan. Whilst not as interesting, already knowing how to daw a cube in code means this could pick up for lost time. |
| AI difficulty imbalance.  (Ai being too easy or hard for practical use) | AI can be fickle and so constant testing throughout should prevent the behavioural foundations from causing imbalances throughout the system. | Selection of difficulty could be a method of turning this issue into a positive outcome, as users may not be competent and the AI will always be balanced against another AI. |

# Specialist resources and support required

As I seek to find the knowledge I require through research of sources and consultation with lecturers, I cannot foresee myself requiring any specialist resources and support.

# Sources and references

Buckland, M. (2005) *Programming Game AI by example*. Wordware publishing.

Jackson, P. (1999) *Introduction to Expert Systems 3rd edition*. Addison Wesley Longman Limited.

Hassani, H. (2012) *How to do final year projects a practical guideline for computer science and IT students* [online]. Bookboon.com [Accessed 01 October 2015].

Kolodner, J. (1993) *Case-Based Reasoning*. Morgan Kaufmann Publishers.

# Monthly project plan

|  |  |  |
| --- | --- | --- |
| October | Proposal submitted by 15/10/2015 | 8 days |
| November | World generation implemented | 21 days |
| December | Research and report complete by 10/12/2015 | 10 days |
| January | Implement A\* pathfinding | 21 days |
| February | Implement combat mechanics and begin combat AI  Finish prototype by 29/02/2016 | 10 days - |
| March | Implement combat AI | 21 days |
| April | Hand-in 14/04/2016 | - |