Introduction to C#

|  |  |  |
| --- | --- | --- |
| **Assessment Task Number:** Task 3 – Design and Implement a State Machine for a Non-Player Character | | |
| **Unit Code(s):** | | **Unit Title(s):** |
| ICTPRG430 | | Apply introductory object-oriented language skills |
| ICTPRG440 | | Apply introductory programming skills in different languages |
| **Items to Submit** | | **Evidence Criteria and Assessment Decision-Making Rules** |
| 1. | State Machine Implementation | A game or demonstration program containing an agent or non-player character implementing an A.I strategy using a State Machine |
| 2. | Design Document | A 2-3 page detailed design of the state machine, including:   * A description of the real-world environment simulated, including any technical parameters * A description of the underlying functionality of the A.I strategy specifying essential settings, states, conditions, and parameters * How the agent interacts with the simulated environment * Difficulty levels and their controls, as appropriate |
| 3. | Record of Feedback | Seek and record feedback on your state machine design.  Record the feedback along with any changes you have made to your design or implementation as a result of the feedback you received.  At least one piece of feedback must be recorded. |
| **Submission Requirements:** | | |
| You will need to submit the following:   * A Release build of each application that can execute as a stand-alone program * Your complete Visual Studio project * A Design Document and Record of Feedback in MS Word or PDF format. These items may be in the same document.   Be sure to remove any temporary build folders (i.e., the Debug and Release folders). Only project files, source code files, and any resource files used should be included in your submission.  Package all files in a single compressed archive file (.zip, .7z, or .rar) | | |

Student’s own self-assessment of outcomes

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Competency | Description | Result |
| [Task 1](#Task1) | Demonstration program contains an agent implementing an A.I strategy using a State Machine. | Agent implements wander/seek behaviours transitionally upon condition of proximity to the ‘player’ agent by use of a finite state machine. |  |
| [Task 2](#Task2) | A detailed design document of the state machine. | A description of the real-world environment simulated, including any technical parameters |  |
| A description of the underlying functionality of the A.I strategy specifying essential settings, states, conditions, and parameters |  |
| How the agent interacts with the simulated environment |  |
| Difficulty levels and their controls, as appropriate |  |
| [Task 3](#Task3) | Record the feedback received and actions taken in response (minimum 1). | 6 topics recorded (combination of feedback given and received) including best practice changes, logic flaw bugs and ‘flow’ optimisations. |  |

1. Task 1
   1. State Machine Implementation

Please refer to attached Visual Studio project.

1. Task 2
   1. Design document
      1. A description of the real-world environment simulated

The program simulates a physical space, inside of which are entities that navigate that space according to various behaviours that they have been assigned, such as wandering randomly or seeking other entities. The space is divided into a bordered ‘map’ of equally sized square tiles, with every tile containing a ‘node’ that defines the parameters of that tile.

Parameters of a tile include whether the tile is traversable by entities or not, visibility over which other tiles may be reached via this tile, and a set of numerical values that influence the degree of priority a given tile will receive when entities consider traversing it. These values include the number of tiles between the entity’s starting position and the tile (‘g’ score), the straight-line distance (irrespective of tile navigability) between the tile and the entity’s desired destination (the heuristic or ‘h’ score), and the sum of these two values (‘f’ score).

* + 1. A description of the underlying functionality of the A.I strategy

**AI strategy chosen (combination wander & seek)**

There are several entities on the map. One entity (the player) is controlled via left mouse click, at which time the player will navigate its way across the map to the tile clicked on if it is navigable, or if not, it will stop moving and remain where it is at the time of mouse click. The behaviour of one non-player entity (the patroller) is determined relative to the location of the player by a finite state machine with two possible states. Movement of all entities is to be calculated according to an A\* pathfinding algorithm, with preference given to the route with the lowest ‘f’ score.

|  |  |  |
| --- | --- | --- |
| State | State-induced behaviour | State activation conditional upon… |
| Wander | Move toward a randomly selected navigable tile on the map | Straight-line distance (irrespective of tile navigability) between the patroller and the player is greater than or equal to 7 tiles |
| Seek | Move toward the player | Straight-line distance (irrespective of tile navigability) between the patroller and the player is less than or equal to 5 tiles |

**Essential settings, states, conditions, parameters**

* The map of node connections must completely and accurately store the navigability information of connecting nodes for tiles to represent a network of paths.
* The 2D Cartesian coordinates of an entity and its goal must be unambiguous to calculate the straight-line distance (the ‘h’ score).
* The pathfinding algorithm which calculates the preferred path of an entity to its goal must accurately calculate the relative ‘f’ score of each tile in each potential path.
  + 1. How the agent interacts with the simulated environment

The patroller is the entity governed by the finite state machine, and it interacts with the simulated environment by

Traversing the map via the map of nodes

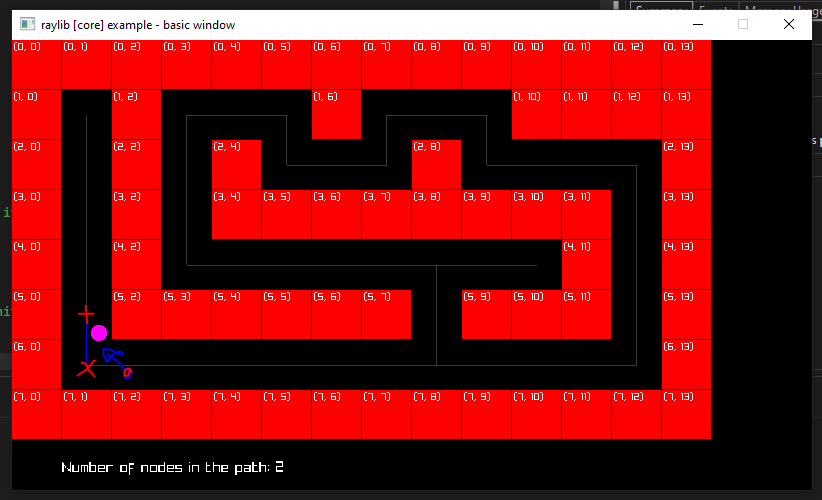
conditions

* + 1. Difficulty levels and their controls, as appropriate

1. Task 3
   1. Record of feedback

Michael Burford and I exchanged feedback on each other’s AI programs back and forth between 20/07/23 and 31/07/23 to try and resolve the following topics:

* Memory cleanup of pointer variables encapsulated by function scope vs pointers with allocated memory, pointer-to-pointer variables that point to a dynamically-sized collection, and the logical Boolean evaluation of pointers. Informed my program-wide approach to use of pointers, no explicit changes. Other explicit changes made after discussion with James Wilmott to only manually delete pointers allocated new memory (best practice changes).
* Agents pathing through walls to one-another’s position when no valid path exists. Fixed by clearing the agent’s path if the path contains only 1 node to navigate to (logic flaw bug).
* Unpredictable 1-node and 2-node behaviour arising from the agent’s pathing decisions occurring at discrete node intervals rather than infinitely small points.
  + Example 1: Instantaneous teleportation. Fixed by clearing the agent’s path if the 0-index of the path does not equal the agent’s starting node (logic flaw bug).
  + Example 2: Three traversable nodes are arranged in a triangle, a corner with points A, B and C, where B is the apex of the corner. If the agent leaves node A to path to node C but is interrupted inbetween B and C, with an instruction to return to A, it will path diagonally rather than crossing over the top of node B. This occurs because the agent already has node B as its ‘current’ node and so has no requirement to path over the top of it before returning to A. Not changed – this is not a bug, this is operating logically correctly, and while appears anomalous in isolation, actually appears much more natural in a sequence of actions taken by the agent when behaving dynamically together with the finite state machine.



* Accidentally building the completed A\* path in reverse; while adding path nodes to a collection, backwards from the end node to the start node, accidentally ‘pushing back’ nodes to the route rather than adding them to the beginning. Fixed by pushing new nodes to the front of the collection (logic flaw bug).
* Differences in results of pathing calculations based on the timing of when the ‘current’ node of an agent is determined, eg before pathing, after pathing, on mouse click, on update. Updated the timing of setting an agent’s ‘current’ node so that the current node is whichever node is closest to the agent at the time of sending an instruction by mouse click (‘flow’ optimisation).
* Run-time vs compile-time calculation differences. Topic for discussion, no explicit changes made.