**AT01 Production Diary**

**1.1.1 Depth First Search Algorithm Research**

**Algorithm Summary**

The **Depth First Search** (DFS) algorithm is an algorithm used for pathfinding purposes with tree/graph data structures. The DFS algorithm at a given node in a tree/graph structure explores the children of a parent node recursively (i.e. depth first), checking the child of the child. The recursive nature of the DFS algorithm continues to explore a path as far as possible before backtracking, repeating the recursive process with the next unvisited node until the goal node is found.

A benefit of the DFS algorithm is its suitability for searching every possible path in tree/graph structures with a few possible paths. A DFS pathfinding algorithm also consumes less memory than other pathfinding algorithms (e.g. Breadth First Search), especially when finding a solution within large tree/graph data structures.

However, the DFS algorithm is not guaranteed to find the shortest path in a given graph structure. The main shortcoming of DFS is it does not check all the siblings of a node, which may result in the algorithm taking a longer path (i.e. the number of nodes processed). The DFS pathfinding algorithm is also a poor choice for tree/graph structures with many paths as it will explore unnecessary paths.

In a video game context, DFS may be used to solve mazes by finding a suitable path/solution in the search. The algorithm may also influence which path to take through a decision-making tree. For example, a chess engine program calculates several outcomes for a board permutation, with the DFS algorithm exploring several series of moves (lines in chess terminology) in depth that may provide the player with a positive outcome.

The DFS algorithm may also be applied in the game Minesweeper with the game progressively revealing adjacent squares after clicking on a safe square until a mine is nearby. If a square was already revealed, the algorithm ignores it to prevent infinite loops.

**DFS Terminology Definitions**

**Pathfinding:**

Pathfinding refers to the methods used to find a path between two nodes, and to find the shortest/most efficient route through a graph data structure through the use of algorithms. Example applications of pathfinding in in a gaming context include NPCs avoiding obstacles while patrolling a location and finding an escape route in a maze.

In terms of the Depth First Search (DFS) algorithm, its pathfinding approach is to recursively explore the children of children in a particular node path as far as possible, backtracking to unvisited nodes if necessary and repeating the process until the goal node is reached.

**Tree:**

A tree is an undirected (no specific direction between connected nodes) type of graph data structure where any 2 nodes are connected by exactly one path and has no cycles (a path that starts/ends at the same node). The nodes are organised in a hierarchal structure with the singular root node branching into two nodes.

A DFS algorithm traverses through a tree structure from the root node, which is a selected arbitrary node in a graph as the start point. The algorithm then explores a path from the root node as far as possible before backtracking.

**Parent:**

In graph structures, a parent is a node that precedes any given node in a graph data structure. A parent node may be identified by its hierarchal position in a graph structure; The node with attached sub-nodes that are lower in hierarchy is deemed the parent node.

Parent nodes are important for the functionality of the DFS algorithm as a parent node is considered a backtracking point for the algorithm. After exhausting its recursive algorithm in a given path, DFS will backtrack to a parent node with unvisited children and reapplies the recursive algorithm in the new path.

**Child:**

In graph structures, a child is a node that descends from a parent node in a graph data structure. A child node may be identified by its hierarchal position in a graph structure; Any sub-node of a given node higher in hierarchy is deemed a child node.

The DFS algorithm recursively checks the children of child nodes of a particular path until it reaches a node with no children, at which point the algorithm backtracks to the parent node with unvisited child nodes to explore the alternate path.

**1.1.3 AI Behaviour Chart**

|  |
| --- |
|  |

**1.1.4 AI Design Reflection**

According to the game brief supplied, the AI should follow the procedure outlined below:

* On game start, the AI will start by moving to the first node in the static list of nodes.
* The AI will then use a depth-first search algorithm to determine which node the player has moved to (target node).
* Whenever the AI reaches its current target node, it will use the DFS algorithm to find a new target node (the current node of the player).
* When the AI collides with the player, the end of the round is triggered, and the game is reset.

The AI will use a trigger collider to determine if the player collided with it, ending the game session and restarting the game shortly afterwards. This will require a dedicated script that handles the game state, as well as code on the AI and player scripts.

With the project file containing a node graph structure, the AI will traverse through these nodes via a DFS pathfinding algorithm to achieve the above outcomes. The following pseudocode will be used:

IF targetNode NOT aiCurrentNode AND targetNode NOT null

currentNode = targetNode

ELSE IF playerTargetNode NOT null AND playerTargetNode NOT aiCurrentNode

currentNode = playerTargetNode

END IF

If currentNode NOT null

Set current direction towards node

Normalise current direction

END IF

And ditto, for the DFS algorithm:

WHILE nodeStack > 0

FOR child in currentNode

IF a visited node has no child AND nodeStack has no child

IF currentNode = playerCurrentNode

RETURN child

Push child to nodeStack

END FOR

END WHILE

RETURN null

|  |  |  |  |
| --- | --- | --- | --- |
| **Feasibility Question** | | **Response** | |
| **Would the proposed implementation work in practice?** | | The implementation of the AI would work in practice; partial implementation was demonstrated during class sessions. With a few tweaks and additions to the pathfinding and AI code, the implementation should work smoothly. | |
| **Has planning been done to support the strategy?** | | The studio project requires some research to be documented in the production diary regarding the Depth First Search (DFS) algorithm, as well as some research into an AI strategy. A brief section of research with human-computer interaction (HCI) devices is also included in the production diary. | |
| **Can the required resources be obtained and integrated?** | | As the working file for the studio project was provided beforehand, the supporting resources are outlined in the “conditions” section of the studio project. Some of the essential resources include: Blackboard learning materials, online access, Unity game engine, an HCI device and the studio head for consultation purposes. | |
| **Strengths** | **Weaknesses** | **Opportunities** | **Threats** |
| The game brief provides a clear idea of how the AI will behave within the studio project. | The AI may not find the optimal (shortest) path towards the player with the Depth First Search pathfinding algorithm. | Adjustable AI speed to improve player engagement and difficulty. | AI may fail to behave appropriately after implementation, requiring technical amendments to fix its behaviour. |
| The AI’s movement patterns will be predictable | The AI requires a root node outside the player’s playable area to begin its pathfinding behaviour. |

A 3-point estimate is detailed below:

|  |  |  |
| --- | --- | --- |
| **3-Point Estimate of Hours Required for Implementation** | | |
| **Optimistic Estimate (O)** | **Most Likely Estimate (M)** | **Pessimistic Estimate (P)** |
| **3** | **5** | **7** |
| **Beta Distribution weighted average: = 5 hours ± 40 minutes** | | |

**1.2.1 Planned HCI Device Integration Summary**

For the studio project, the chosen human-computer interaction (HCI) device is an **XBOX One** controller. The controller is compatible with XBOX One, XBOX Series S/X and other devices with Windows, macOS, Linux and Android operating systems. The controller is an updated version of the XBOX 360 controller with minor design changes to the shape of the triggers, analogue sticks, shoulder buttons and rumble motors. Originally launched in 2013, three revisions were made to the XBOX One controller from 2015, 2017 and 2020 respectively to include 3.5mm jacks, Bluetooth support and USB port changes.

The specific controls to be implemented into the Unity studio project is the control #8 **left analogue stick Horizontal (X axis) / Vertical (Y axis) movement**. The movement of the analogue stick will link to the directional UI widget when moved in the respective direction (Up, Down, Left, Right) to control player movement. The movement will be of a binary nature, with the player moving at a predetermined speed regardless of the amount of analogue stick lock.

|  |
| --- |
|  |

**1.2.2 C# Event System Summary**

Events are used in producing interactive media as a notification to a program, allowing it to invoke a method’s functionality after receiving a particular input (e.g. button clicks, menu selections in a GUI).

A potential industry-standard method for implementing the XBOX One controller functionality to the Unity studio project is to set the mapping with the integrated input manager. An event is then created in the Player script with references the mapped inputs, listening for player input before invoking the correct function to move the player in a specific direction.

**1.2.2 Unity GUI Library Review**

Two types of Unity GUI libraries are **uGUI** and **IMGUI.**

* The Unity User Interface (**uGUI**) is an older UI system in Unity which uses GameObject interactions. Its primary purpose is to develop UI for games/applications at runtime. uGUI allows for the arrangement, positioning, and style of various user interface elements, supporting rendering and text functionality. Applications of uGUI include the creation of a UI canvas, positioning, sizing and animating of UI elements as well as handling user interactions.
* The “Immediate Mode” GUI (**IMGUI**) is the code-driven GUI system used in Unity, which is useful for programmers and debugging. The IMGUI system operates on calls to the OnGUI function on any applicable script, drawing user interface elements on each frame. Applications of IMGUI include creating in-game debugging displays and tools, custom inspector UI and editor windows/tools to extend the functionality of Unity.

For the Unity studio project, the uGUI library will be used for production to create, position and animate the UI widget as described in the game brief. A possible debugging option for the UI widget may manifest as an IMGUI window, improving debugging efficiency by outputting messages that confirms user input.

**1.2.3 UI Widget Example Overviews**

|  |
| --- |
| **UI Widget Example #1 – Shadow Fight Arena (Mobile)**  The fighting game uses a directional wheel UI widget on the left side to move the character in 8 directions. Players can either drag the centre wheel to the respective direction or tap the direction to move. In the image above, the HUD is highlighting the right direction for the player to press as part of a tutorial. |
| **UI Widget Example #2 – PUBG (Mobile)**  The battle-royale game provides users with several control schemes to play the game, with one configuration providing a movement UI widget on the left side of the screen. Players drag the centre wheel to the respective direction to move, with simulated analogue input depending on how far the player drags in a specific direction. In the image above, the HUD customisation shows the movement wheel on the left side of the screen, also displaying the drag range to enable the player to sprint. |

**1.2.4 UI Widget Paper Prototype**

|  |
| --- |
| **Logo  Description automatically generated with medium confidence**A simple design of a prototype UI widget, with the squares representing the 4 directions the player can move in the Unity studio project – Up, down, left and right.  4.  Default colour of squares is white.  2.   * Green flash if player can move * Red flash if player cannot move   3.   * Mouse (LMB Click) * Keyboard (WASD) * Controller (Left analogue stick)   1.  Functionality - Widget script  Input – Player Script (Input.GetAxis)  Mappings – Input Manager   1. The direction will flash green if the player can move in that direction and red if the player cannot move. The flash occurs immediately on keyboard/analogue stick input and lasts for 0.5 seconds before returning to the white colour. Until the player finishes moving to a node, the UI widget will ignore additional inputs. 2. To implement the above functionality, the Widget script will include code that sets all squares to white as default. The Player script will include code that moves the player to the respective direction via Input.GetAxis horizontal/vertical. Inputs to be registered by the UI widget will be set in the Input Manager for the peripheral devices 3. The keyboard mappings will be WASD for up, left, down and right respectively, while the controller’s analogue stick will follow the mapping set in the Input Manager. The widget will also respond to left mouse button input. Receiving input from any peripheral device will invoke the respective directional movement and flash colour functions. |

**2.1.1/2.2.2 Testing Log**

*Please add rows as required.*

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case Description | Expected Results | Actual Results | Success? |
| Testing for correct node hierarchy with user input | The player should move to each node (except the root node) in the correct direction as outlined in the node hierarchy. | The top node row was behaving unexpectedly, with inverted vertical movement. The player could move to the root node. | No |
| Testing revised node hierarchy from previous test case | The player should move to each node (except the root node) in the correct direction as outlined in the node hierarchy. | The player moved to each node (except the root node) in the correct direction. | Yes |
| Testing for widget functionality (Mouse click) | The UI widget should load on runtime, default to white and flash green/ holding red when input is registered. Clicking on the widget will move the player in the correct direction. | The UI widget performed all runtime operations without any problems. | Yes |
| Testing input mappings set in the Input Manager (Keyboard and Controller) | Both keyboard and controller inputs must be set correctly, registering input and moving the player in the correct direction. | The key mappings for both peripheral devices performed as expected. | Yes |
| Testing AI behaviour and related scripts (Collision, movement speed, DFS pathfinding) | The AI must travel to the player’s previous node via Depth First Search, be able to collide with the player and have a working speed property. | All AI properties and related scripts were performing as expected, except for the DFS pathfinding algorithm. | No |
| Testing corrected DFS pathfinding | The AI must travel to the player’s previous node via Depth First Search. | The DFS pathfinding algorithm was performing as expected. | Yes |
| Testing GameManager script (Load/Restart game) | The game should end when the AI collides with the player and restart the game after a brief pause. | The GameManager script functionality was performing as expected. | Yes |

**3.1 Final Checks**

|  |  |
| --- | --- |
| **Final Checks** | **Confirmed** |
| * AI pathfinding (using the DFS algorithm) has been successfully integrated | Y |
| * Game over conditions have been successfully implemented | Y |
| * Appropriately compatible with Google Chrome web browser | Y |
| * Appropriately compatible with Mozilla Firefox web browser | Y |
| * Appropriately compatible with Windows | Y |
| * UI widget responds to relevant keyboard inputs | Y |
| * UI widget responds to relevant mouse inputs | Y |
| * UI widget responds to relevant controller inputs | Y |
| * UI set to scale with a full HD resolution (1920x1080) | Y |

**3.1 AI Evaluation**

|  |  |
| --- | --- |
| **Strengths** | **Weaknesses** |
| The AI movement is smooth and seamless when moving to the player’s previous node. | The initial predetermined speed of the AI was too slow, requiring changes to the speed property to be more challenging for the player. |
| The AI consistently follows the DFS pathfinding algorithm. | The AI does not find the shortest path to the player’s node if the player crosses a specific set of nodes (Node hierarchy dictates the order of DFS). |
| The AI is able to collide with the player and invoke the Game Over sequence of the GameManager script. | The speed of the AI may be too fast for some players, no easy way to adjust the speed stat for both the AI and player. |

**3.2 Required Amendments**

|  |  |
| --- | --- |
| **Amendment** | **Outcome** |
| Add the missing line of code which sets the value of targetNode to the return result of the DepthFirstSearch algorithm. | Added the following code: targetNode = DepthFirstSearch() to the Enemy script.  Testing revealed the AI was following the player as expected with the DFS algorithm. Project considered complete, build Windows and WebGL builds and submit. |

**3.3 Final Client Sign-Off**

*Insert a screenshot of your email communications with the client, providing evidence of their endorsement to finish the production of the project.*

# References

Brown, J. (2019, July 9). *PUBG Mobile Controls Guide – Layout Differences, Customization, and Profiles*. Retrieved from Fanbyte: https://www.fanbyte.com/guides/pubg-mobile-controls-guide-customization-profiles/

GeeksforGeeks. (n.d.). *Depth First Search or DFS for a Graph*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

Hammerly, A. (2017, March 30). *Depth First Search & Minesweeper*. Retrieved from Thagomize: https://thagomizer.com/blog/2017/03/30/depth-first-search-minesweeper.html

happycoding. (n.d.). *Pathfinding: Depth-First Search*. Retrieved from happycoding.io: https://happycoding.io/tutorials/libgdx/pathfinding#depth-first-search

hurna. (n.d.). *Depth First Search (DFS)*. Retrieved from hurna.io: https://hurna.io/academy/algorithms/maze\_pathfinder/dfs.html

*Shadow Fight Arena*. (2020). Retrieved from GameUIDatabase: https://www.gameuidatabase.com/gameData.php?id=1048

tag104367. (2017, May 07). *Xbox one controller mapping [SOLVED]*. Retrieved from Unity Answers: https://answers.unity.com/questions/1350081/xbox-one-controller-mapping-solved.html

Wikipedia. (2022, May 13). *Depth-first search*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Depth-first\_search