CI517: Game Engine Fundamentals

AI Pathfinding

Technical Documentation

13th May 2020

Prepared by:

Oliver Andrew Lee

14813961

University of Brighton

Contents

YouTube and GitHub Link

A video demonstration of the project can be found as a unlisted YouTube video by following this link: https://youtu.be/yuRWKuW4ljY. Should it also be desired, the project source code can be found on GitHub through https://github.com/Torakon/xcube. This GitHub repository has a record of every commit made so far and is up to date as of the submission of this documentation.

Implementation

Chosen Engine Aspect

The system chosen for this assignment is the AI subsystem, specifically the pathfinding aspect. This was selected from the wide range available due to its perceived challenge and the opportunity it presents to learn from it. Commenting and documentation within the Visual Studio project is performed according to documentation provided by Doxygen (2020) with aspects of the Engine that were present in and that remain unchanged from the provided version excluded.

An AStar search algorithm was chosen for the pathfinding of the NPC entities within the Test Game as, despite its space complexity, it tends to have a better performance than some algorithms such as Best First search by using heuristics. Initial research into the algorithm lead to a copy of an article by Patrick Lester (2005) which was used to gain an understanding of the steps involved in an AStar search.

The implementation in this project uses three classes: '*AIController*', '*AStar*' and '*Node*'. Each '*Node*' instance stores information relevant to a single tile including its location on the provided map, it's relative cost from the start node of the search (G) and estimated cost to the goal node (H). It also stores a reference to a Parent '*Node*' for backtracking through a path and possible movements in the four cardinal directions.

The '*AIController*' is initially responsible for creating and initialising the relevant variables of each '*Node*' and storing it in a vector of vectors of shared_ptr '*Node*'s. This is performed based on a map passed in by the Test Game (in this case, a vector of Rectangles). Once this representation of the map is generated it can give paths to any Entity that calls one of its two '*givePath*' functions. The first instance provides a path from one Entity to another and handles AI behaviour such as random patrolling if the player isn't in range whilst the other one provides a Path from the Entity to a specified Point2 location.

This Path is obtained through calling '*AStar*' where the code for the actual algorithm is located. This code starts with the current '*Node*' in an '*Open*' vector before switching the lowest cost '*Node*' in this collection to a '*Close*' vector. From there it will check each edge '*Node*' from the newly switched one, if these ones are not already in the '*Open*' collection it will add them there and make the current '*Node*' the parent of these. If it finds any of these '*Node*'s already in the '*Open*' collection but as part of a longer path it will recalculate the G and H costs and make the new Parent the current '*Node*'. A '*Node*' is considered as part of a longer path if it's cost (F = G + H) is larger in the '*Open*' collection than that of its counterpart with the more recent calculations.

It will, however, simply skip the '*Node*' if it is instead found inside the '*Close*' collection. These steps will loop until the goal '*Node*' is switched over to the '*Close*' collection or when the '*Open*' collection is empty (which would mean that there is no path available). Once it has a confirmed path it will then backtrack through the parent '*Node*'s whilst adding their coordinates to a Point2 vector; the final path it will return.

Since the AI can have a sight distance value where zero or less will signify an unlimited sight, the search algorithm has an additional termination point where the depth constraint has been exceeded and therefore will only provide a partial path to the goal.

Steps were taken to ensure that the implemented aspects were developed to have multiple use cases available to them, so they are not specifically tied down to the Test Game. Any code relevant to AI pathfinding and behaviour is self-contained although it does still expect certain information to be passed to it from the program that chooses to make use of it. Because the '*AIController*' handles behaviour such as random patrolling, it is also expected that this program be a game, should this program make use of the added '*Entity*' section of the engine there should be little to no problems as behaviours and sight are initialised as false by default. There is also the ability to just use point to point pathfinding if so desired.

Vectors were chosen as the primary data structure used within the AI pathfinding code due to their general similarity to Arrays, which cannot be dynamically declared in C++. Lists seemed like the preferred choice, however since the size of the primary vector should not change it should not need to resize often and therefore provide better performance.

Outside aspect scope

Beyond the scope of the chosen engine aspect, additional engine aspects were either created or edited. One of these is the basic '*Entity*' system that both the player and NPC's now use. It is not specific to the Test Game and is stored on the engine side. This class stores the entities position (Collider, Texture Display and actual Position), behaviour flags and traversing path if applicable.

It was created in order to ease the creation of multiple NPC instances alongside being able to easily assign each one its own path and traversal progress whilst including very basic behaviour and individual textures if needed. Whilst this class works very well with the '*AIController*' functions, the pathfinding can be used without it when using the point to point pathfinding.

An aspect that received minor modifications was the '*eventEngine*'. However the extent of these modifications were minor, reaching only to an addition of two additional keys to the already available selection.

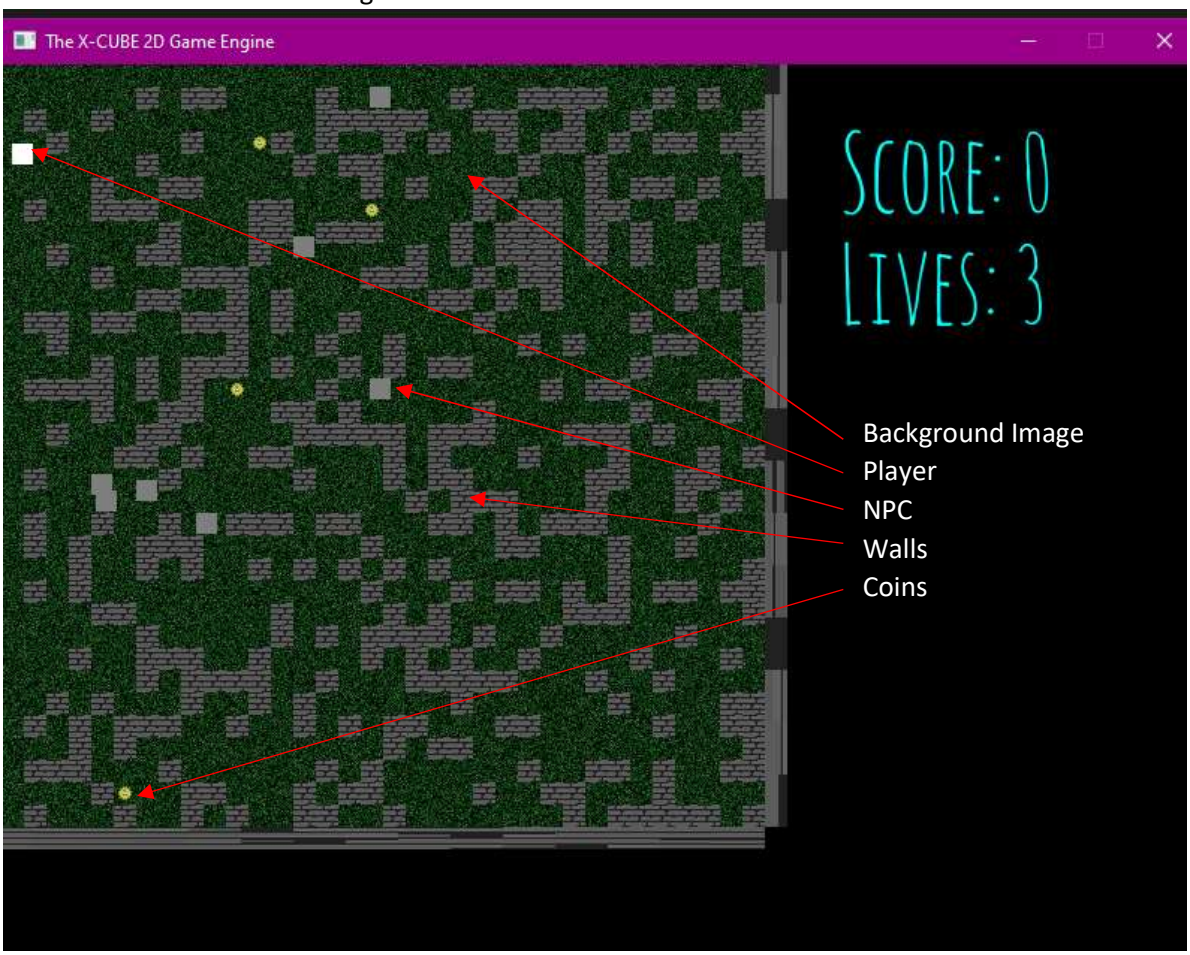
Another file that received modification during the development of this project was the '*CMakeLists*' file. This was because early on in the development of the Test Game it was decided that an mp3 format was desired for the sound played when a coin was collected, however the CMake file at the time did not have a line to include the relevant dll file (*libmpg123-0*). Whilst one could copy the file over to the correct folder, the addition to the CMake file means that the project gets built with the correct library transferred automatically.

Game Demo

Assets

This section will provide an overview of the assets utilised within the Test Game demo. All visual assets were created independently with exception to the custom font. Sound files were also obtained from an asset repository. All assets utilised in the Test Game demo that were not created specifically for this project are referenced in the '*References*' section at the bottom of this document. '*sndFailure*', '*sndCoin*' and '*sndCoin2*' were acquired from a pack (Hanna, D. 2016) listed under a Public Domain license (Creative Commons). '*sndTheme*' (Kauffman, C. 2018))is being used under the CC BY-SA 4.0 license (Creative Commons). Finally, the font '*Amatic*' (Adams, V. 2012) is used under the Open Font License (SIL. 2007).

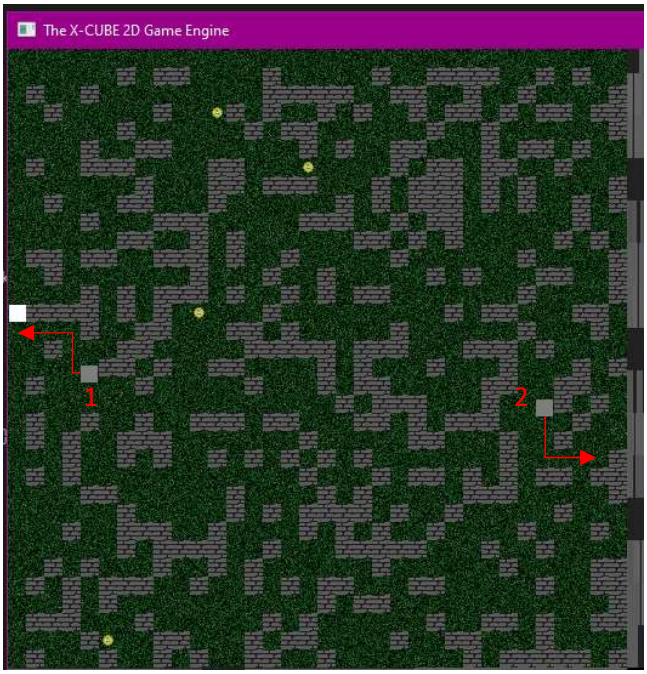
The Test Game opens up on an initial Menu where '*sndTheme*' first begins playing on loop. Both loaded fonts are utilised here with '*Amatic*' taking the place of the main text whilst a smaller version of the originally provided font is shown in the buttons. By this point, all used assets are already loaded via the Resource Manager.



The visual assets can be seen in the annotated screenshot above, during a level within the game. First drawn is the background image, beneath anything that is drawn subsequently. It then draws the NPC and Player entities by getting the texture pointer stored in the entity object alongside the display area.

The render code then cycles through the stored wall locations and draws the wall texture in their place. This allows for the NPC Entities to spawn 'under' the walls with those that do not have a patrol behaviour only emerging when the player is near. The coins are rendered last so that they can be seen above everything.

Aspect



The next image shows the actual AI component with arrows representing their movement. NPC one is within range of the Player and will pathfind to directly to them. On the other hand, NPC two is not in range and therefore is pathfinding to a random point within its sight distance.

Whilst there is technically only one '*AIController*', it can give out multiple paths since each NPC stores it's own path for it to follow.

Within the Test Game, the spawned NPC entities are recorded within a vector with each one moved incrementally along their given path each update tick. Entity movement is performed by setting their actual X and Y coordinates and then updating their Display rectangle and collider bounding box accordingly. This keeps the texture render and the collision detection in line with where the Entity actually thinks it is.

A new path is only calculated and provided to an Entity when it requests it, in this case when the Entity has made a certain percentage of progress through its current path.

Critical Review

One aspect of the designed subsystem that was created to a good standard is the method in which the map data is stored, in particular the edge system. Before this was implemented, whilst searching for a path, the search would get the pointer to each edge '*Node*' before checking if they were walkable before dropping those that were not. However, with the edge system in place the search now simply checks the current '*Node*' for moveable connections and only gets the pointers to edge '*Nodes*' that can be moved to.

Another area that is implemented well is the ability to check if a valid path exists. In the Test Game, this is used to ensure that all key pickups spawn in a location that is reachable by the player due to the fact that the map is procedurally generated. It's a relatively short piece of code that could have many use cases. An example of which could be a player shutting an interactable door which would have the affect of cutting off access to AI and preventing them from chasing the Player therefore telling them to return to their previous behaviour.

The map code that takes the given information and converts it from a single vector of rectangles to its new data structure of '*Nodes*' is also very succinct and performs well. It generates a rectangle in each Tile coordinate and checks to see if it intersects with the rectangles stored in the one dimensional vector, it will create an instance of a '*Node*' and set the relevant walkable parameter. Once this is done it will set the edges. This representation of the entire map provided performs for a user defined map size, along the X and Y axis independently provided the tile size fed into the function fits neatly into it. A suitably descriptive exception is thrown otherwise.

All projects have aspects and areas that they could see improvements with and this project is no exception. One area that could benefit from some extra attention is the Data Structures used within the actual search algorithm. Whilst vectors are suitable for the map representation, the often changing vector sizes within the search algorithm would have benefitted more from using a list structure, at least in terms of efficiency. Each time a vector resizes it must copy over each element to the new vector in a similar way as a resizing hash table. An alternative solution may also be present in the ability to reserve memory for the vector in question, similarly reducing the need for resizing.

As it stands, the backtracking in the search algorithm is in need of improvement. It currently gets the current path and puts it into a vector data structure before 'flipping' it into another. This is not ideal but is in place as a workaround due to an issue involving the vector insert function causing the Test Game to hang. A potential fix to this is actually outside of the AI subsystem; the way in that the Entities process the path could be reversed so that it works from the end of the structure instead of the beginning. This would mean that only one vector utilising the push back function would be needed. On the other hand, the possibility of lists could be explored; unlike vectors, lists have a push front function.

In regards to the '*Node*'s, a prospective expansion would be to change the 'walkable' boolean to an integer, thus allowing for different types of terrain and traversal speeds. An example of this would be grass with the default weight of one and deep mud with a weight of two whilst walls keep the weight of 0. This could be incorporated into the search algorithm by adding it into the F calculation so that AI prefer to path around if possible (and faster) with different 'weights' added to different tiles.

References

Adams, V. (2012). *Amatic*. Available at: https://www.fontsquirrel.com/fonts/amatic Last Accessed [11th May 2020]

Creative Commons. (Unknown Year). *Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)*. Available at: https://creativecommons.org/licenses/by-sa/4.0/ Last Accessed [12th May 2020]

Creative Commons. (Unknown Year). *CC0 1.0 Universal (CC0 1.0) Public Domain Dedication*. Available at: https://creativecommons.org/publicdomain/zero/1.0/ Last Accessed [12th May 2020]

Doxygen. (2020). *Doxygen Manual: Documenting the code*. Available at: http://www.doxygen.nl/manual/docblocks.html Last Accessed [12th May 2020]

Hanna, D. (2016). *8-Bit Sound Effect Pack (Vol. 001)*. Available at: https://opengameart.org/content/8-bit-sound-effect-pack-vol-001 Last Accessed [8th May 2020]

Lester, P. (2005). *A* Pathfinding for Beginners*. Available at: http://csis.pace.edu/~benjamin/teaching/cs627/webfiles/Astar.pdf Last Accessed [11th May 2020]

SIL. (2007). *SIL Open Font License (OFL) 1.1*. Available at: https://scripts.sil.org/cms/scripts/page.php?item_id=OFL_web Last Accessed [11th May 2020]