TNM095 - Artificial Intelligence for Interactive Media
October 31, 2022

A* Algorithm to Achieve Personalities in Attacking Enemies of a Tower Defense Game

Tilda Hylander[1], Jonathan Andersson[2]

Abstract
The aim of this project was to simulate different types of behaviours and personalities using the A* algorithm. This was achieved in a tower defense game where the heuristic function and neighbouring nodes, of the pathfinding algorithm, vary depending on the enemy type. The game was implemented in Unity with a free asset providing an A* pathfinding algorithm. Despite common use as a optimization algorithm, the A* algorithm can also be used to create more complex types of behaviours of attacking enemies in games.

Video: https://www.youtube.com/watch?v=34j1xTd0obg

Authors
[1] *Media Technology Student at Linköping University, tilhy349@student.liu.se*
[2] *Media Technology Student at Linköping University, jonan270@student.liu.se*

Keywords: Interactive AI, A*, Pathfinding, Unity

Contents

1. Introduction

Optimization algorithms such as the A* algorithm is a vastly popular method for solving problems and puzzles in games. Often, A* is used to find the shortest path according to some heuristic function towards solving a problem. This project instead aims to replicate some personality traits among the attacking enemies in a tower defense game by changing the nodes that are considered as connected and their distance for different enemies.

This project has been created by combining two courses: *TNM095 - AI for Interactive Media* and *TDDD23 - Design and Programming of Computer Games*. The report will focus on aspects related to the implemented AI in the project as part of the course TNM095 while more gameplay related features have been submitted as a part of the course TDDD23.

2. Background

This chapter covers the background and related theory needed to understand the implementation of the game.

2.1 Game Idea

The idea for this game was to create an interactive tower defense game with attacking enemies that portray different personalities. Traditional tower defense game can be quite static where the attacking enemies follow a pre-defined path and the player is only allowed to place towers in between waves. By implementing AI for the attacking enemies it is possible to create a more exciting and entertaining tower defense game. The path-finding algorithm can be modified to simulate different types of personalities among the enemies such as brave, smart and coward enemies. The player will then have to learn the behavior of all the types of enemies and adapt their defense to suit the enemy abilities.

2.2 Pathfinding

Pathfinding is the concept of finding a path between a start node and a goal node. Most pathfinding algorithms work on graphs defined by a set of vertices with edges connecting them. A grid-based game map can be considered as a graph with each tile being a vertex and edges are drawn between tiles that are considered as adjacent to each other.

There are different types of path-finding algorithms such as Dijkstra's Algorithm, Greedy Best First Search and A* Algorithm. Dijkstra's Algorithm works by visiting nodes in a graph starting with the starting node. It then repeatedly evaluates the closest not yet evaluated node, adding its nodes to the set of nodes to be examined. It expands outwards until it reaches the goal node. The Greedy best first search works in a similar way except that it was a heuristic estimate of how far a node is from the goal node. It works faster than Dijkstra's algorithm but does not always provide the best path. A* algorithm combines these two approaches and can guarantee to find the best path [1].

2.3 A* Algorithm

A-star algorithm (also referred to as A*) is one of the most successful search algorithms to find the shortest path between nodes. It uses information about path cost and also heuristics to find the best solution. A* achieve optimality (which means that it is guaranteed to find the best solution) and completeness (which means that if a solution exists, the algorithm is guaranteed to find it), two valuable property of search algorithms [2].

The A* algorithm makes use of two queues. One queue is named *open* which contains nodes that are still valid candidates when searching for an optimal path and the other queue is named *closed* that contains nodes which have already been evaluated and therefore should not be selected for evaluation again [3]. Each node is evaluated according to:

$$f(n) = g(n) + h(n) \tag{1}$$

Where $h(n)$ represents a heuristic function which estimates the cost from the start node to current node n. It needs to be admissible, meaning that it should never overestimate the cost, while $g(n)$ represents the actual cost. The open list of nodes are sorted based on the estimate cost. Since the cost is never overestimated, the optimal solution to the problem can be found according to Definition 1 [4].

2.4 Pathfinding with Emotion

The idea to implement personality-like behaviour sparked when reading the report *Pathfinding with Emotion Maps* of Anja Johansson and Pierangelo Dell'Acqua. It uses findings within psychology and neuroscience together with emotion maps to model a more varied and natural agent-like behaviour in agent pathfinding. Their objective was to create interesting behavior for games [5].

3. Method and Implementation

This chapter will present the method for implementing the game. The game was implemented in Unity.

3.1 Environment

The playing field of the game is represented by a two-dimensional grid containing tiles, see Figure 1.

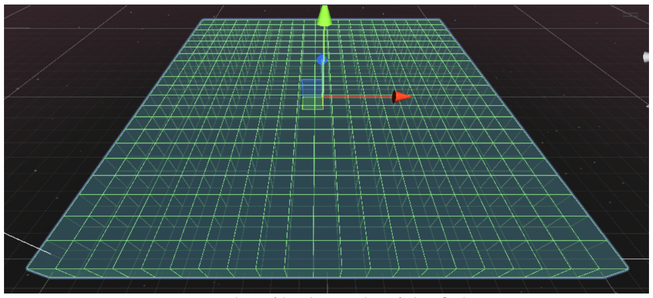


Figure 1. The tile-based grid of the map.

Tiles each have an environment type: grass, water or mud, see Figure 2. The environment type of a tile is generated procedurally using noise but the noise parameters are fixed per map giving similar, but not entirely equal map results each time the game is played. The goal tile of the map is represented by a castle.

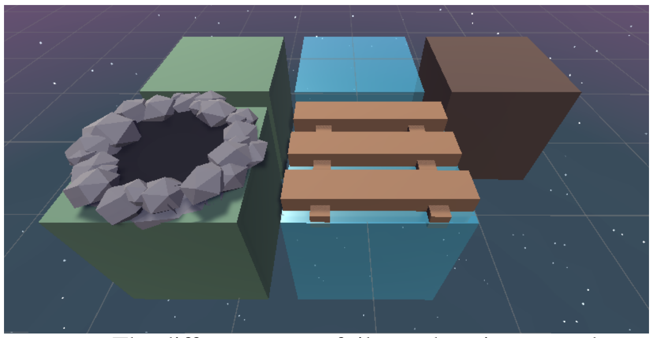


Figure 2. The different types of tiles and environmental structures that can occupy a tile.

Tiles can also be occupied by structures that can affect whether the given tile is walkable or not, see Figure 2. Walls and tunnels can occupy tiles of type grass which makes the tile non-walkable for most enemies. Bridges can occupy tiles of type water which makes the tile walkable for all enemies. The number of tunnels for a given map is set as a fixed integer and the placement of tunnels is random. Bridges are placed by first finding all grass-tiles that lack a walkable path for the general enemy to the goal and then applying the A* algorithm to find a path to the goal placing bridges where necessary.

Each tile also holds a *danger-factor* property. When a defensive unit is placed, all tiles with a certain proximity to the it have their danger-factor increased, which makes it possible for some enemies to evaluate a *safe* path to the goal. The proximity depends on the range of the placed defensive unit, see Figure 3.

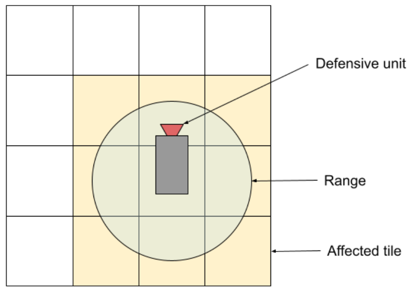


Figure 3. The tiles affected by the range of the placed defensive unit

3.2 A* Pathfinding Solution

For the implementation of the A* algorithm the *Ultimate A* Pathfinding Solution* [6] was used, which is available as a free Unity Asset. This A* path-finding solution contains a generic class designed to fit any situation if the class is supplied with a method that calculates a heuristic approximation of distance between two nodes, and a method that takes an instance of a node and returns all the connected nodes and their distances.

3.2.1 Heuristic Distance Function

The heuristic distance function takes two instances of nodes and calculates the distance between them. This function is implemented as the Manhattan distance between two nodes in the grid, see Equation 2.

$$d = |x_1 - x_2| + |y_1 - y_2| \tag{2}$$

The heuristic distance function is the estimate of the distance $h(n)$ between the current node n and the goal node and should always give the accurate value or underestimate. The Manhattan distance is therefore a correct estimation of the distance in all cases except if the tile is occupied by a tunnel, then the distance could be shorter if there is a tunnel close to the goal. In this case the heuristic distance function will return a constant value.

3.2.2 Connected Nodes And Distances Function

The connected nodes and distances function takes an instance of a node and returns its connected neighbours and their distances. This function is specified for each type of enemy, thus producing different behaviours. The aim was to simulate different personalities by changing the way the enemies calculate their path to the goal.

3.3 Element

Each enemy has its own specialized path-finding algorithm that creates their specific behaviour. Elements mainly affect the connected nodes function of the path-finding algorithm. By changing this function it was possible to control which tiles where considered in the calculation of an enemy's path toward the goal tile. A tile in the grid is connected to all of the 4-connected neighbouring tiles, see Figure 4, that are considered as walkable for that specific enemy. Enemies with different elements consider different tiles as walkable, but in general the tiles that are accessible are bridge-tiles, mud-tiles and grass-tiles that are not occupied by a wall.

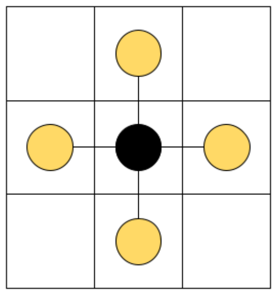
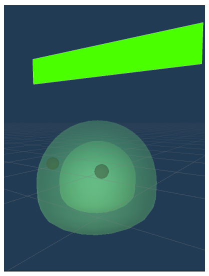
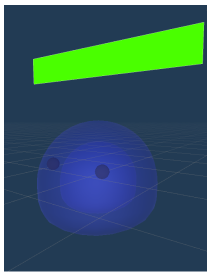


Figure 4. 4-Connected neighbour tiles

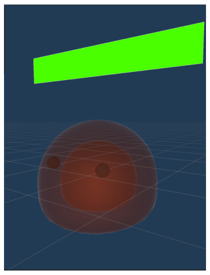
Different elements are visualized in the game using different colors, see Figure 5.



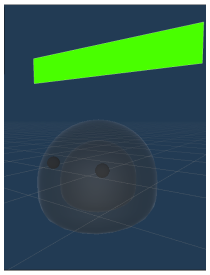
(a) An enemy with neutral element type



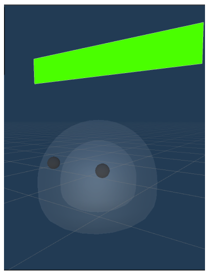
(b) An enemy with water element type



(c) An enemy with water element type



(d) An enemy with stone element type



(e) An enemy with ghost element type

Figure 5. The visual differences between enemies with different elements

3.3.1 Neutral

Enemies with the element *Neutral* act as the general case, and consider bridge-tiles, mud-tiles and grass-tiles that are not occupied by a wall as accessible when calculating their path.

3.3.2 Water

Enemies with the element *Water* consider all general accessible tiles but also water-tiles. This means that the calculated path to the goal may include water-tiles.

3.3.3 Mud

Enemies with the element *Mud* act as the general case such as Neutral, but the speed of the enemy is increased to the double when walking on a mud tile.

3.3.4 Stone

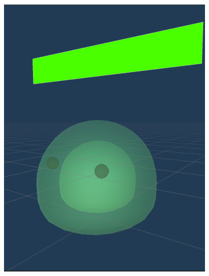
Enemies with the element *Stone* consider all general accessible tiles but also tunnel-tiles. This means that enemies with this element consider tunnel-tiles as walkable as well as considering that all tunnel tiles are connected. By walking to a tunnel tile the stone enemy can *teleport* to another tunnel tile anywhere on the map.

3.3.5 Ghost

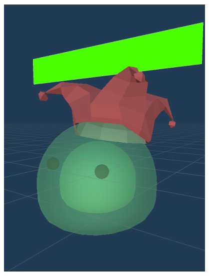
Enemies with the element *Ghost* consider all general accessible tiles but also all grass-tile that are occupied by a wall. This means that enemies with ghost element can walk through walls.

3.4 Personality

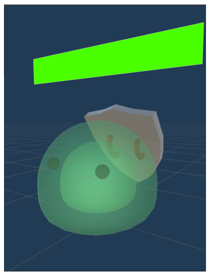
One important decision making factor for the implemented enemies is their assigned personality. A personality of an enemy was simulated by modifying the step-cost for the connected nodes and distances function in the path-finding algorithm. Different personalities are visualized in the game using different accessories, see Figure 6.



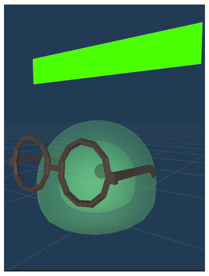
(a) An enemy of personality type simple



(b) An enemy of personality type coward



(c) An enemy of personality type brave



(d) An enemy of personality type smart

Figure 6. The visual differences between enemies with different personalities

3.4.1 Simple

The most basic type of personality is the personality labeled as *Simple*. Simple enemies will pick the shortest path according to Manhattan distance, which implies the number of tiles they have to traverse to reach the goal. Being the most basic, the simple personality type is introduced first in the game.

3.4.2 Coward

Another personality is *Coward* which differ from Simple enemies by not always picking the shortest distance in terms of tiles traversed. Instead, they look at the danger-factor property of the tiles in order to find the safest path to the castle by avoiding defensive units as much as possible. Coward enemies also have less health but move faster than Simple enemies.

3.4.3 Brave

Brave enemies act the same as Simple enemies in terms of pathfinding, but differ in terms of slower movement and higher health points.

3.4.4 Smart

Smart enemies are similar to Cowards in that they also avoid tiles based on danger-factor, however their multiplier for the danger-factor is not as large as the one used by Cowards. Smart enemies are the only enemy personality that have a varying heuristic distance based on the tile environment. Smart enemies of element type Mud will prefer tiles of type mud since they will be able to move faster in the mud, while smart enemies of different element types will avoid mud since they will move slower in the mud.

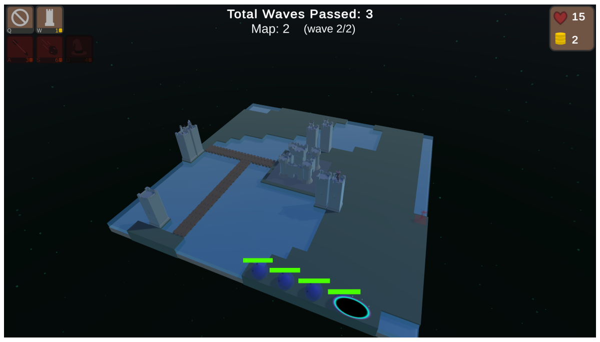
3.5 Placing wall structures and defensive units

In order to place defensive units, the player first have to place walls. Defensive units can then be placed on top of walls. Each time a wall or unit is placed, enemies will re-evaluate their path since the state of the game has changed. This will sometimes result in enemies choosing a completely different path to the one they were currently following when a wall or unit is placed.

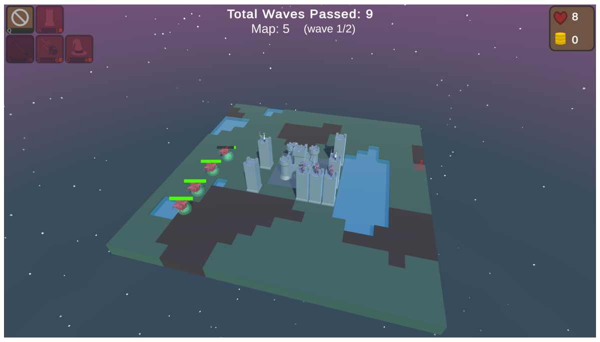
It is important that the player may never be given the opportunity to place walls in such a way that enemies may be unable to reach the castle since this would mean that the A* algorithm would fail in finding a valid path. This is ensured by disallowing the player from placing walls where a neighbouring tile to the wall would lose its path to the castle. This ensures that the player cannot build closed structures and that all walkable tiles at the edge of the map will have a walkable path to the castle.

4. Result and Discussion

The end result is a playable tower defense game where the player is faced with waves of enemies that behave differently depending on personality and element type, see Figure 7. In the game, personalities and elements affect the pathfinding of the enemies. The personality type affects the chosen path for an enemy while the element affects the available connected tiles when choosing a path. This means that the player needs to adapt their strategy to the type of enemy that is spawned on the map, leading to a more challenging game than if all enemies behaved the same way. The behaviour of the enemies is also interactive, which means that they might change paths in response to an action made by the player.



(a) Enemies of element type water heading for the castle. The planned path will cross the water tiles.



(b) Enemies of type coward moving to the castle while also avoiding dangerous defensive units.

Figure 7. Examples of gameplay images

A* Algorithm is suitable for this type of assignment since it will not visit every tile of the map but only the most relevant ones (based on heuristics) and will end if the goal tile is found. It guarantees to find the best path and always find a path if there is one which gives some security in this type of task, since the game could crash if no path is found. The maps are constructed to always have a least one path from all possible spawn tiles to the goal tile.

The personalities of the attacking enemies are graphically visible using color and accessories. Some of the personalities are very well noticeable such as the Coward enemy which will walk as far as possible just to avoid the range of the defensive units, which portraits the characteristic of a coward in a good way. Other personalities are less obvious such as the Brave personality which is represented by a slower speed and higher health, which could be a little harder to understand as it is similar to the Simple.

The Smart personality acts very similar to the Coward personality in most cases. In order to properly convey a Smart personality, more unique heuristic traits could be added such as avoiding big groups of enemies.

The elements of the enemies are more obvious as they change the walkable nodes, which seems to work in a good and clear way.

Parts of the game could be optimized further. For example the path evaluation when placing walls could use Greedy Best First Search Algorithm instead of A* for the sake of efficiency since finding the optimal path would be irrelevant in this case. Other optimizations could be to improve the re-calculation of the path of the enemies by only changing the path if it is influenced by the player's placement. Another idea for future work is to create more types of enemies, for example a personality and element type that would allow an enemy to climb walls and attack defensive units.

5. Conclusion

While commonly used as an optimization tool, the A* algorithm can also be used to simulate more complex behaviours and decisions of enemies by changing the heuristic function and which nodes are considered as connected. This can be exploited in a game setting, thus producing interesting and fun gameplay.

References

[1] Amit Patel. Introduction to a*. `http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html`. (accessed: 01.11.2022).

[2] Baijayanta Roy. A-star (a*) search algorithm. `https://towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb`, 09 2019. (accessed: 31.10.2022).

[3] Dede Kurniadi, Asri Mulyani, and Rizky Safta Maolani. Implementation of pathfinding algorithm in sundanese land history educational game. In *2021 2nd International Conference on Innovative and Creative Information Technology (ICITech)*, pages 145–150, 2021.

[4] Pierangelo Dellacqua. Tnm096 lecture 3 - informed search. `https://dellacqua.se/education/courses/tnm096/material/slides/2022/lec3.pdf`, 04 2022. (accessed: 01.11.2022).

[5] Anja Johansson and Pierangelo Dell'Acqua. Pathfinding with emotion maps. `https://www.researchgate.net/publication/302263711_Pathfinding_with_Emotion_Maps`, 01 2012. (accessed: 01.11.2022).

[6] Aoiti. Ultimate a* pathfinding solution. `https://assetstore.unity.com/packages/tools/ai/ultimate-a-pathfinding-solution-224082`, 06 2022. (accessed: 31.10.2022).