Pathfinding Worksample Project - Code and Art made by Dennis Stockhaus

Description

The implementation explores the basics of path-finding and path-following in a 2d-sapce as well as some State-Based Behavior for a AI-character. When starting the application a map and a ai-character is created.

- •The map gets a randomly generated set of walls and the user is asked to choose a destination by clicking somewhere on the map.
- •The ai-character uses a A* pathfinder to determine the shortest path from it's current position to the target.
- •A animation displaying the pathfinders search-steps is shown.
- •The ai-character walks from its current position to the target position.
- •The ai-character displays wether or not a path was found by setting it's animation to either 'celebrate' if successfull or 'cry' if not.
- •Then a new maze gets generated and the process starts over.

Framework and Techniques

The project is made using Visual Studio 2019 and is written in C++. SFML (Simple and Fast Multimedia Library) is used tas the base engine for graphical base-components and rendering.

Key Components

Pathfinder

A class that contains the function 'getPath' which recieves a representation of map, start position and target position as arguments. It uses a A* algorithm with a manhattan-distance heuristic function to determine the shortest path from current position to target, returned as a vector of integers.

ProblemSolver

The "ai-character" which is tasked to find the shortest path from it's current position to a target position and walk to target position. The ProblemSolver contains a animationSprite component to display and manage animation-changes and a aiController that manages the states for the character.

ProblemSolverStates

A Finite State Machine is used to manage the different states for the ProblemSolver. it uses

a AIController component to manage all states with a AIState base-class to decouple the behaviors. The states for the ProblemSolver is the following:

• AskForProblemState – Generates a random set of walls and asks the user to pick a destination by clicking somewhere on the map.

•FindPathState - Makes an asynchronized call to Pathfinder.getPath and checks each update if a path has been found. When the pathfinder is ready it changes state.

•DrawPathProcessState - Animates the A* pathfinder search process in two steps. first it animates the tiles the algorithm investigated/visited by drawing a colored tile at a specified frame rate. Then it draws the path using the same principle.

•FollowPathState - Using a pointer to the ProblemSolver's animation component, it moves the character along the path generated by the pathfinder. at the meantime it updates the characters walkcycle-animation to one of four directions matching the current movement direction.

•ShowResultState - Depending on the result from pathfinder's success or failure, one of two animation's is played. 'celebrate' animation if successfull or 'cry' animation if not. The result is displayed for 3 seconds and then a new FindPathState is initialized.

Maze

A object that contains a TileMap, represented by a 2D grid containing tiles for Walls and Floors.

WorldState

The state of all objects that should be drawn and or updated is managed through a Service Locator. WorldState provides a global point of access to the service. Some of the services that WorldState provides is the following:

•AddNewObject - Adds a new object with the common base class 'WorldObject'
•RemoveObject - Removes a specified object from all containers in WorldState
•GetAllDrawableObjects - Returns the full list of Drawable Objects. (used by render-loop)
•GetAllUpdatableObjects - Returns the full list of Updatable Objects. (used by update-loop)
•GetObject - Returns a specific 'WorldObject' specified by a object-id. (used for global communication between objects)

References

[AI for Games, Third Edition, Ian Millington]

[Game Programming Patterns, Robert Nystrom]

[Game Programming Algorithms and techniques]