**Whose a good boy. Dog walking sim**



**Setup**

Please refer to readme text file

**Project Intention**

The intention of this project is to show how a robust AI can be created through pathfinding algorithms, graph theory, finite state machines and object orientated programming.

<https://en.wikipedia.org/wiki/Pathfinding>

<https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)>

<https://en.wikipedia.org/wiki/Finite-state_machine>

<https://en.wikipedia.org/wiki/Object-oriented_programming>

regardless of the users skill level from beginner to computer science bachelor, by playing the game, reading this documentation and or reviewing the code the user can gain an a better understanding and appreciation how a basic AI can be programmed and modified.

**How to play**

**direct input**

use direction keys to move your character

to call your dog press space. your dog will now follow you around the map.

press s to make your dog stay

press left ctrl to play chase with your dog. You dog will now run away from you.

press p to pick up the ball

press t to throw the ball. the dog will now stop whatever it is doing and run to the ball and bring it back to you.

**indirect**

Different animals respond differnetly depedning upon what kinds of animals and items are close by.

for example squirells will be indiffernt to people but will run and hide when a dog is near.

in future versions this will be expaned.

eg. ducks can be fed by humans but will flee if dogs are near.

dogs can play with other dogs.

greyhounds can chase rabbits et.

**How the AI works**

The AI can be broken down into two main parts Finite State Machine and pathfinding algorithm. Modification of the AI behavoir between animals is acheived through Object Orietnated programming.

**Finite state machine (FSM)**

Below is an example of super basic state machine for coin operated turnstyle.



Each animal has their own distint FSM, with each state directing what there behavior is. the animals state will change depnding upon direct input from the player or environment, switching them to the correct state and behavoir for the context.

For example if a dog is in stay state they will not move, when a player throws a ball this will switch the dog to fetch state where they will run to the ball and pick it up, once the dog has picked up the ball they will switch to return to player state and so on.

another example is the squirrel. the squirrel will start in the sniff state where they will randomly move around and not move more than 5 blocks from a tree. If a dog comes with 10 blocks of the squirrel, the squirrel will switch from sniff to flee state where they will run to the nearest tree and hide.

Below is the dog's FSM showing the different states it has and what triggers it to move between states and change behavoir.



Below is the squirrel's FSM showing the different states it has and what triggers it to move between states and change behavoir.



**Pathfinding Algorithms**

For pathfinding the A\* search algorithm is currently used. This algorithm uses graph theory to calculate the shortest path from one node to another. The psuedo code is described below.

// A\* Search Algorithm

1. Initialize the open list

2. Initialize the closed list

put the starting node on the open

list (you can leave its f at zero)

3. while the open list is not empty

a) find the node with the least f on

the open list, call it "q"

b) pop q off the open list

c) generate q's 8 successors and set their

parents to q

d) for each successor

i) if successor is the goal, stop search

successor.g = q.g + distance between

successor and q

successor.h = distance from goal to

successor (This can be done using many

ways, we will discuss three heuristics-

Manhattan, Diagonal and Euclidean

Heuristics)

successor.f = successor.g + successor.h

ii) if a node with the same position as

successor is in the OPEN list which has a

lower f than successor, skip this successor

iii) if a node with the same position as

successor is in the CLOSED list which has

a lower f than successor, skip this successor

otherwise, add the node to the open list

end (for loop)

e) push q on the closed list

end (while loop)

The game is made up of tiles that the animals and object sit on. Mathematically these tiles can be considered as connected nodes on a graph, or node network, making it possible for the A\* path finding algorithm to work.

**how the code works**. when an animal needs to move from a start point to an end point it, the code will remove all nodes from the graph that have a barrier sitting on them, before passing the graph to the A\* function. Therefore A\* will avoid the barriers because their nodes are not included in the graph.

This allows the animals to avoid different barriers. eg when a dog need to find a path all nodes with water and fences on them will be removed. But when a duck needs to find a path only nodes with fences will be removed. Allowing different animals to intereact with the terrain differently.

**Please note**

Each node currently given the same wieght so the algorithm will pick the path based around barriers based on number of nodes it takes to get from the start to finish and will not take into account the terrain. Eg an animal will make no distinction between a paved road or rough terrain. This will be address in future editions by adding a terrain weight to nodes and updating the algorith to take this into account when calculating the nodes g - score. Differnt animals can also have differnt weights for differnt terrains. eg. a human will prefer a paved path over grass, but a dog will weight grass and paved road equally.

Currently animals can not move diagonally, this is because only nodes above, below, left and right are considered neighbors, so A\* does not consider them an option. future versions will include diagonal movements in pathfinding.

**Data Structures.**

Game grid is a 2d array the stores if a node is a barrier or empty. this data structure is used by the path finding algorithm to find efficient paths from on point (start) in the game grid to another (end). this grid does not store anything about the player or npc character only if a position on the grid has a barrier or not.

below is how the game grid stores the nodes

[

[barrier,barrier, barrier,barrier,barrier, barrier],

[barrier, start, empty, empty, empty, barrier],

[barrier, empty, empty, empty, empty, barrier],

[barreir, barrier, barrier,barrier,empty, barrier],

[barrier, empty, end, empty, empty, barrier],

[barreir, barrier, barrier,barrier,barrier, barrier]

]

**Object Orientated Programming**

The modification and change of animal AI is efficeintly achieved through the use of classes and inheritence.

A new animal whose class inherits from another animal will also inherit it's FSM. by adding, removing and or overloading(modifying) the functions that make up the inherited FSM, the new animal can have a distinct AI.

**Animal**

**Dog**

**Squirrel**

**Release schedule**

**Good Boy**

Animals:

player controlled human

dog

squirrel

items

ball

features

basic animal pathfinding

dog will, follow, stay and play chase and fetch

ball can be thrown by the player

squirrels will flee and hide from dogs

**Gooder boy**

Animals

Ducks

ducks

features

players can use the computer mouse to throw the ball

advanced pathfinding, weighted graph search

diagonal movement

3rd release

Animals

Terrier

features

player can pat dog

basic game loop. dog can get tired, once taken home the game is won

player can have multiple dogs

dogs can play together

4th release

**animals**

human NPCs

Grey hound

features

grey hounds can be used for coursing rabbits

5th release

sounds

NPC humans can have dogs

6th release

animals

pitbull

swan

feauters

pitbull can kill other dogs if player lets them get too close

swan will live near ducks, swan will be nice to humans but hyper aggressive to dogs and can kill smaller dogs if player takes them too close.