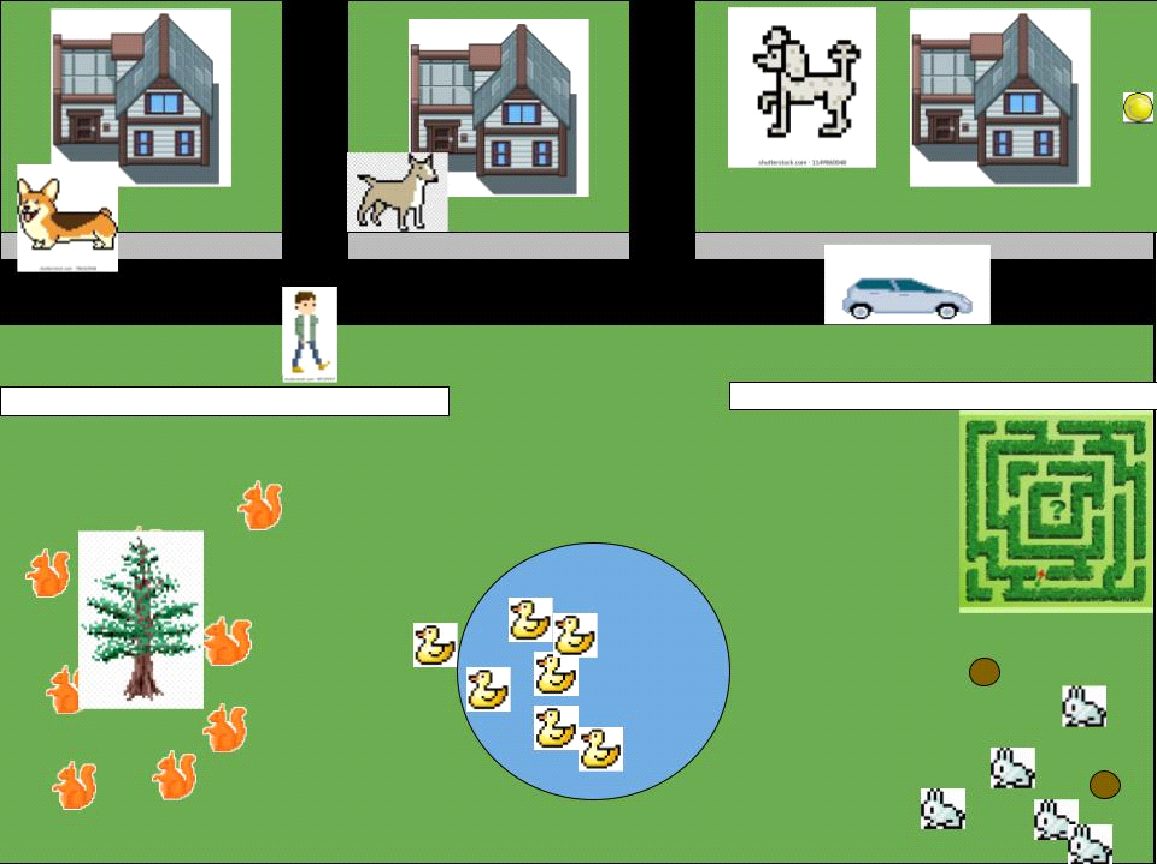
**Whose a good boy. Dog walking sim**



**Table of Contents**

[Setup](#f6meashh8953)

[Project Intention](#6uw4smydv9jh)

[How to Play](#5r0kqr3q6k7h)

[How the AI works](#yus604c6jm5w)

[Finite State Machine](#glxwsfhu7rm6)

[Pathfinding Algorithms](#n2rwfxubqjct)

[Data Structures](#lj52dn9r2epy)

[Object Orientated Programming](#8ero0ezhe0sn)

[Updates Schedule](#rg9meinrlk5g)

**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 user’s skill level, by playing the game, reading this documentation and or reviewing the code the user can gain a better understanding and appreciation for 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. Your dog will now run away from you.

Press p to pick up the ball

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

**Indirect Interaction**

Different animals respond differently depending upon what kinds of animals and items are close by.

For example squirrels will be indifferent to people but will run and hide when a dog is near.

In future updates these interactions will be expanded.

For example:

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

-dogs can play with other dogs.

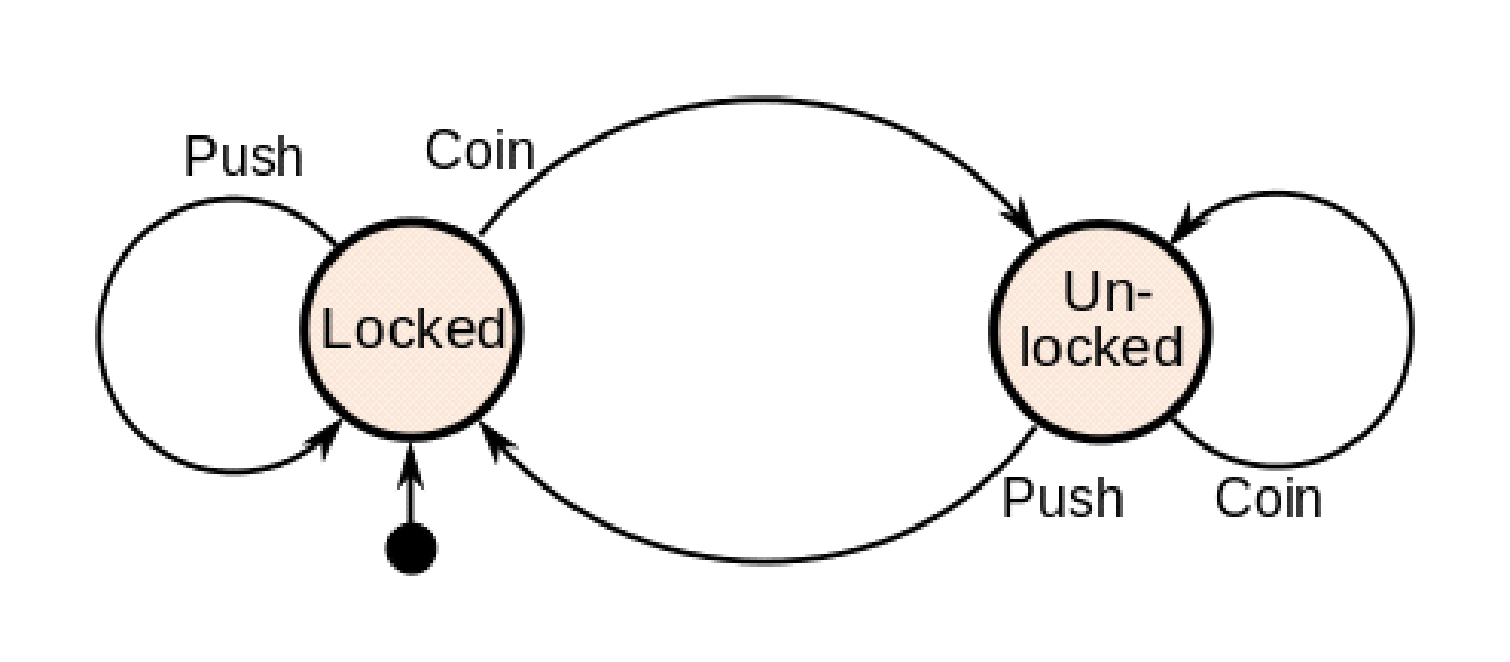
-greyhounds can chase rabbits et.

**How the AI works**

The animal AI can be broken down into two main: the **Finite State Machine** and **pathfinding algorithm** specifically A\* algorithm. While modification of the AI behavior between animals is achieved through **Object Oriented Programming**.

**Finite state machine (FSM)**

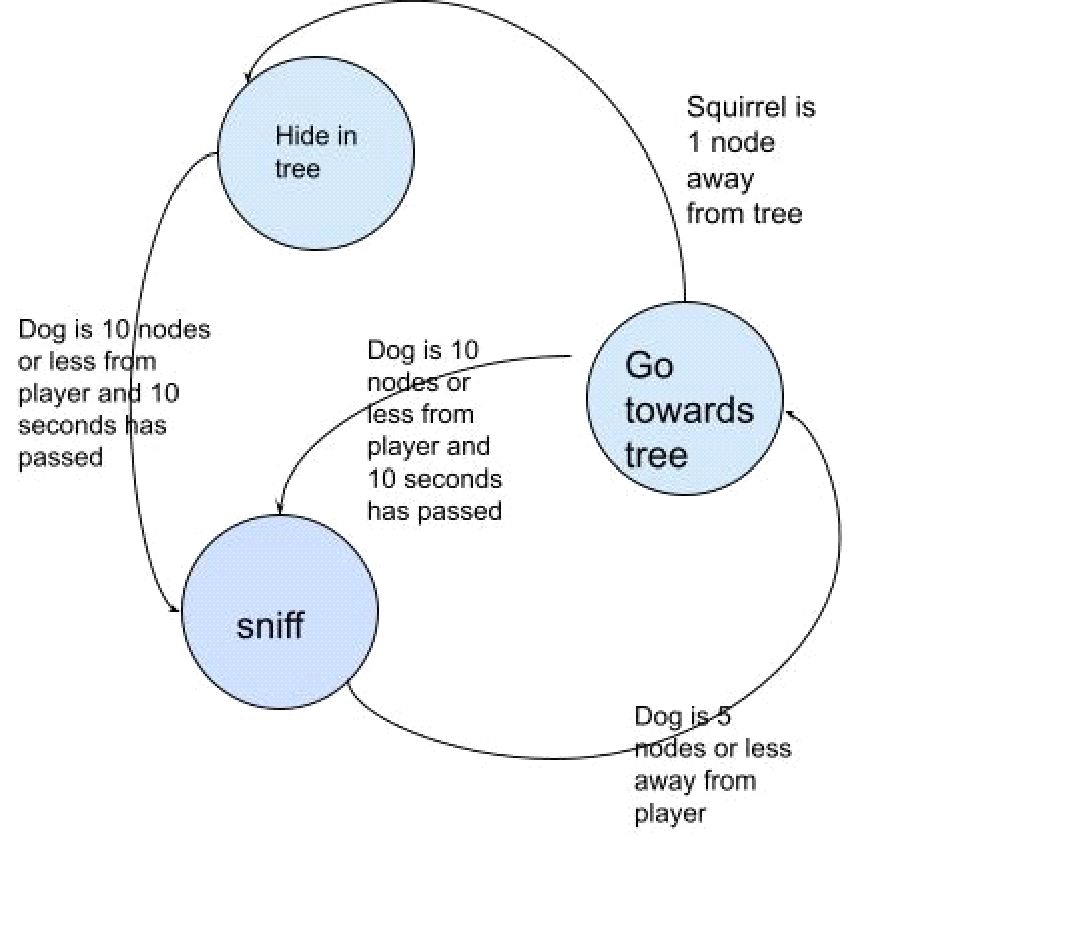
Below is an example of super basic FSM for coin operated turnstile.



FSM allows the program to change the animals behavior based upon inputs or the context of what is happening in the game environment. This allows the animal to exhibit a behavior that is correct to the context of what is happening in the game.

For example if the player throws the ball the dog will switch from 'stay' state to a 'fetch' state or if a dog goes too close to a squirrel the squirrel's state will change to 'flee'.

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



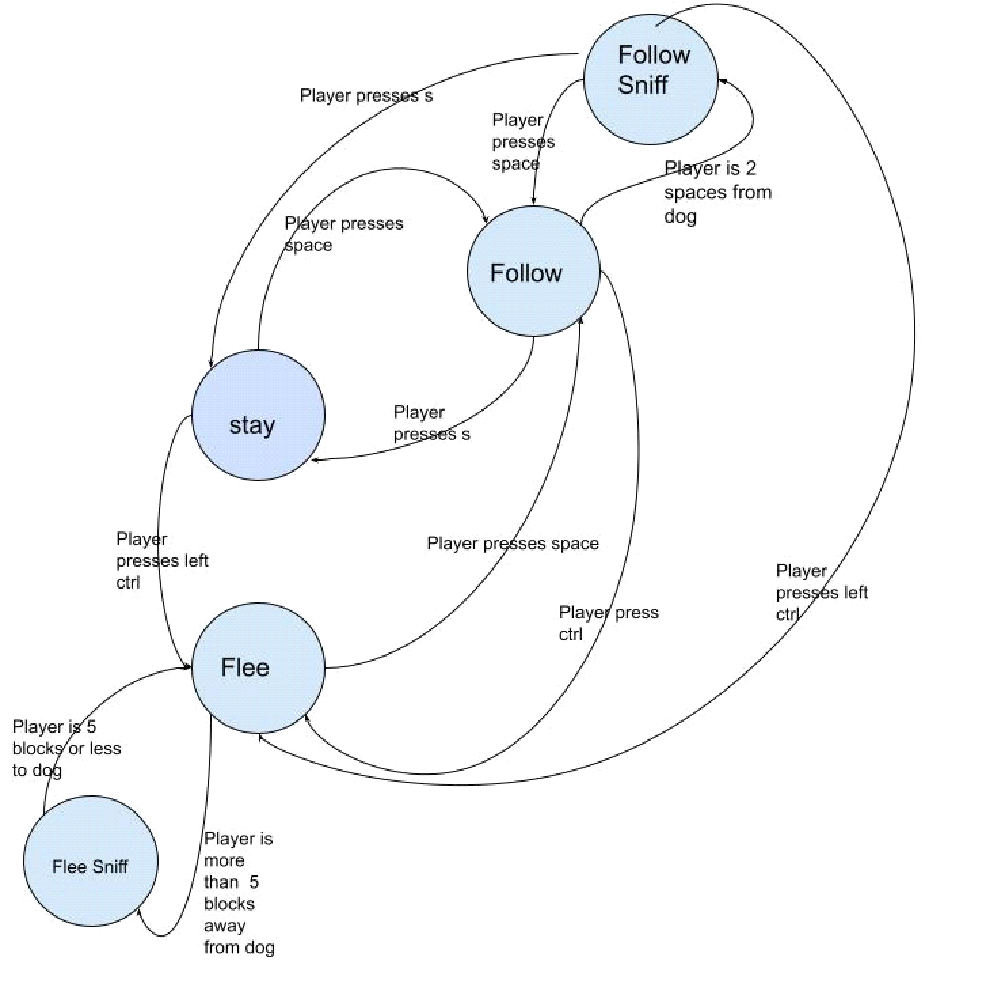
**Squirrel States**

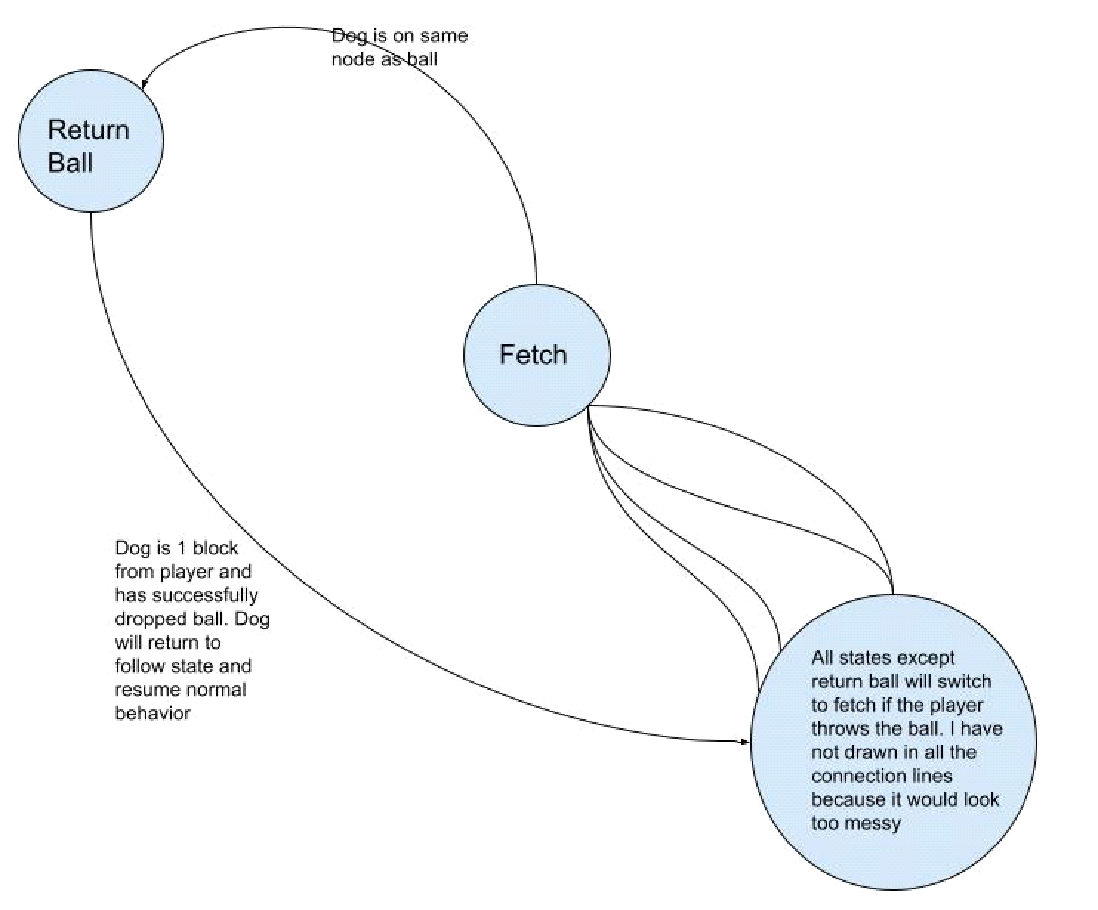
Sniff: squirrel will randomly sniff around trees

Go towards trees: squirrel will get a path to closest tree and start moving their

Hide in Tree: squirrel will stay in tree node and not move

Below is the dog's FSM showing the different states it has and what triggers it to move between states and change behavior. It has been broken up into two sections to improve readability.





**Dogs States**

Stay: dog will not move

Follow: dog will get a path to the player and begin moving to the player

Follow sniff: dog will randomly sniff around the player

Flee: dog will get a path at least 5 blocks away from the player and begin moving there

Flee sniff: Dog will randomly sniff around but will not randomly move onto node 5 or less from then player

Fetch: dog will get a path to the ball and begin moving their, once at the ball it will pick up and change state

Return ball: dog will get a path to the player and begin moving their. Once at the player it will drop ball and change state

**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 pseudo 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

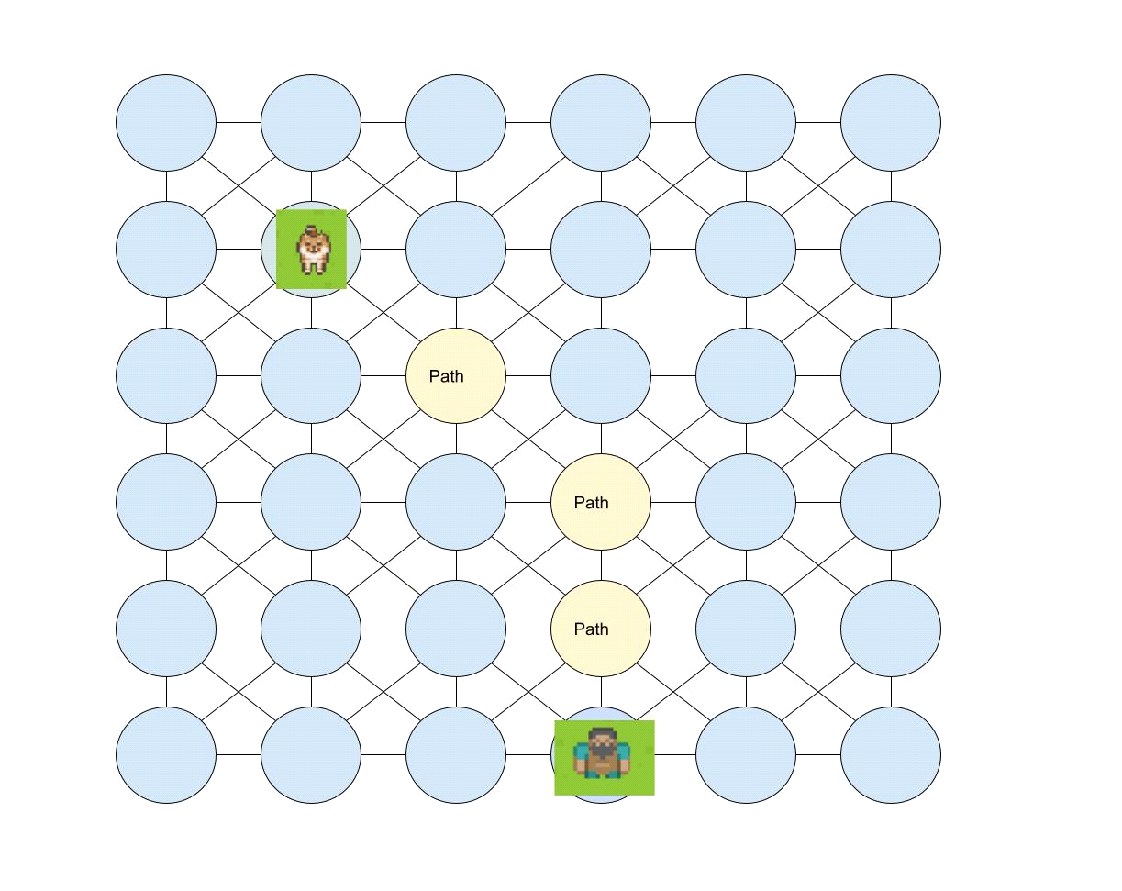
otherwise, add the node to the open list

end (for loop)

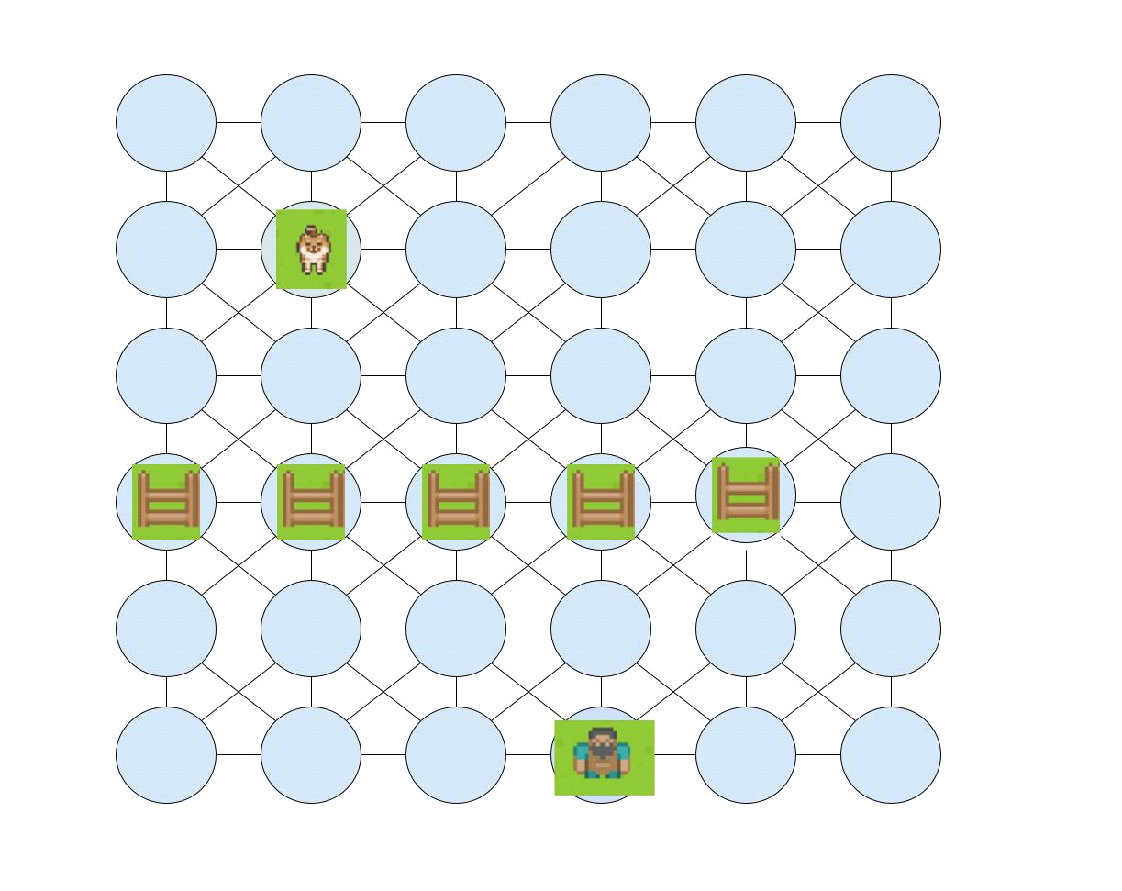
e) push q on the closed list

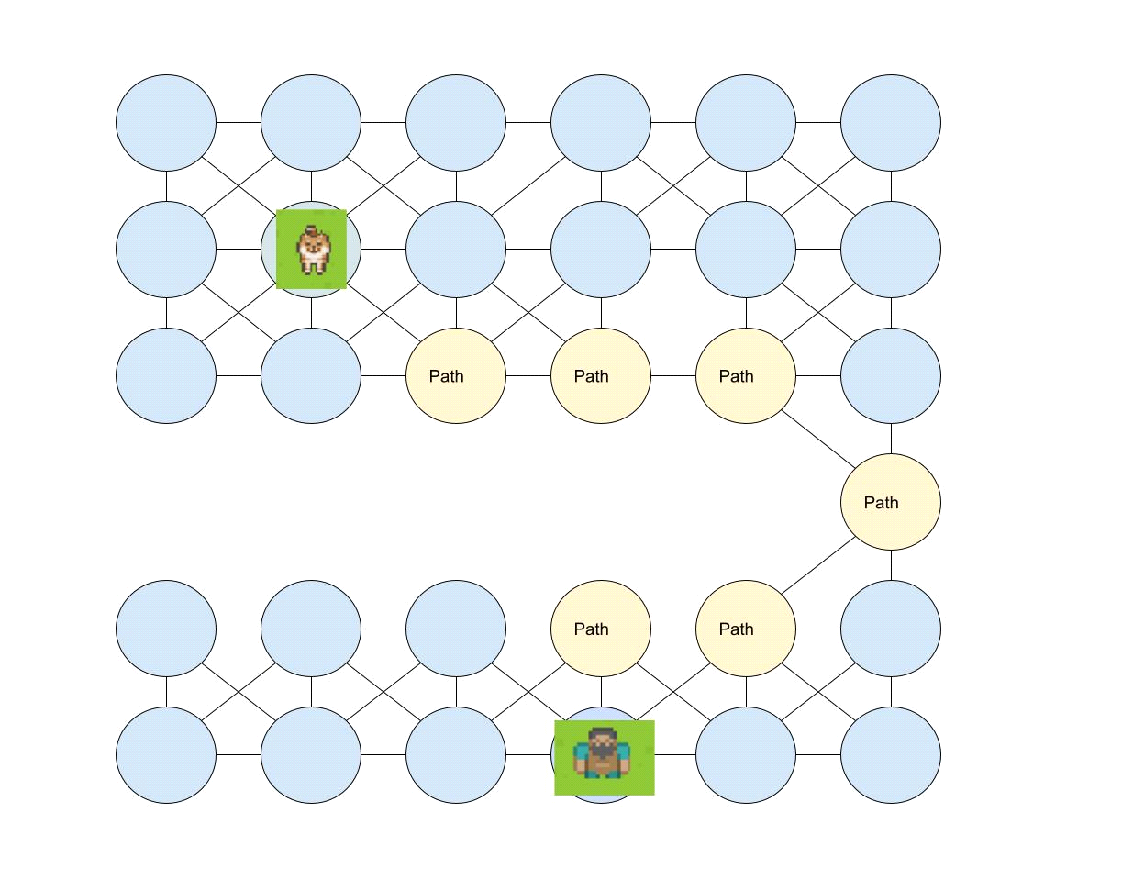
end (while loop)

The game is made up of tiles that the animals and objects 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.



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.

Graph has obstacles that the dog needs to avoid.

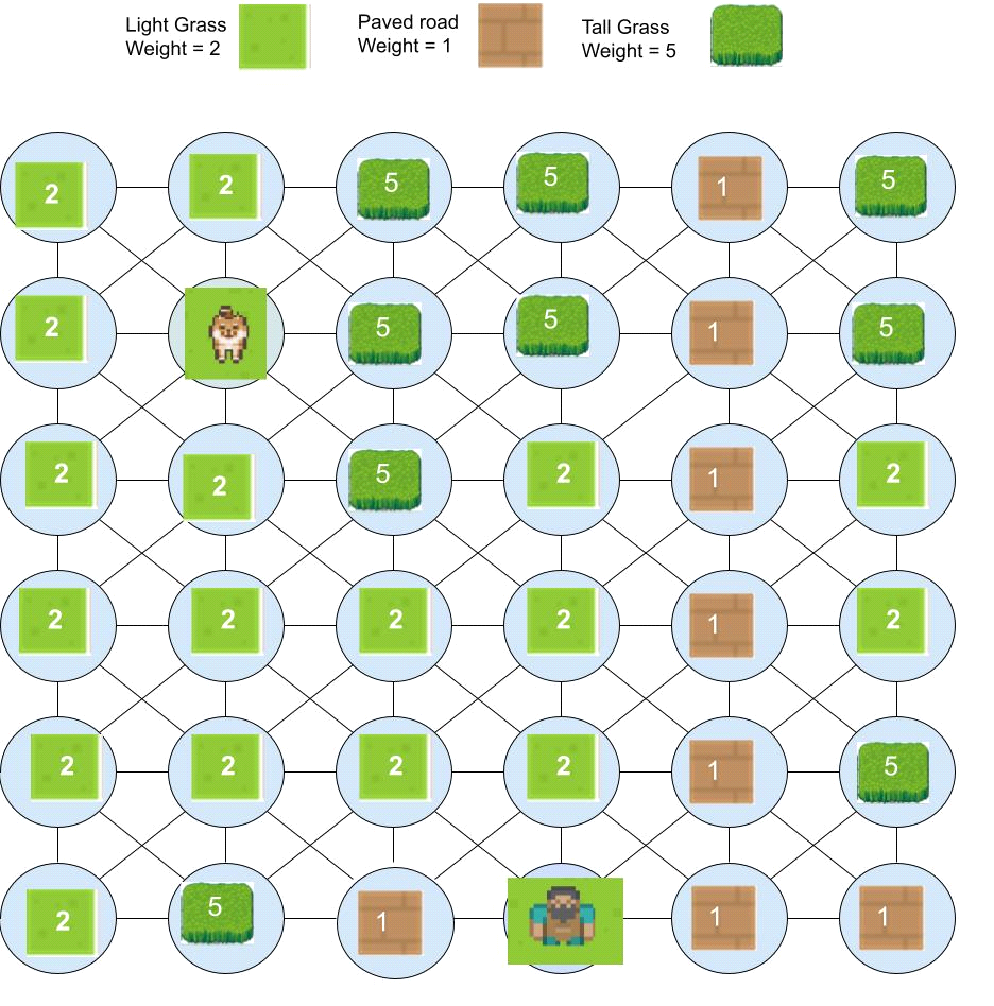
Remove nodes with obstacles, A\* then calculates path

This allows the animals to avoid different barriers. eg when a dog needs 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 interact with the terrain differently.

**Please note**

All nodes have the same weight 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 addressed in future updates by adding a terrain weight to nodes and updating A\* to take this into account when calculating the nodes g - score. Different animals can also have different weights for different terrains. eg. a dog will prefer a paved path over grass, but a squirrel will weigh grass and paved road equally.

below is an illustration of the weighted graph can take into account the terrain, the terrain weights will allow the A\* algorithm to select node for path based on both distance and terrain.



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.**

Nodes - nodes hold information used by A\*for generating a path such as the node's neighbors and weight and

Game grid holds all the nodes in a 2d grid replicating the actual game map

[

[node, node, node, node],

[node, node, node, node],

[node, node, node, node],

[node, node, node, node]

]

A\* returns a list of all the nodes making up the path

[node1,node2,node3]

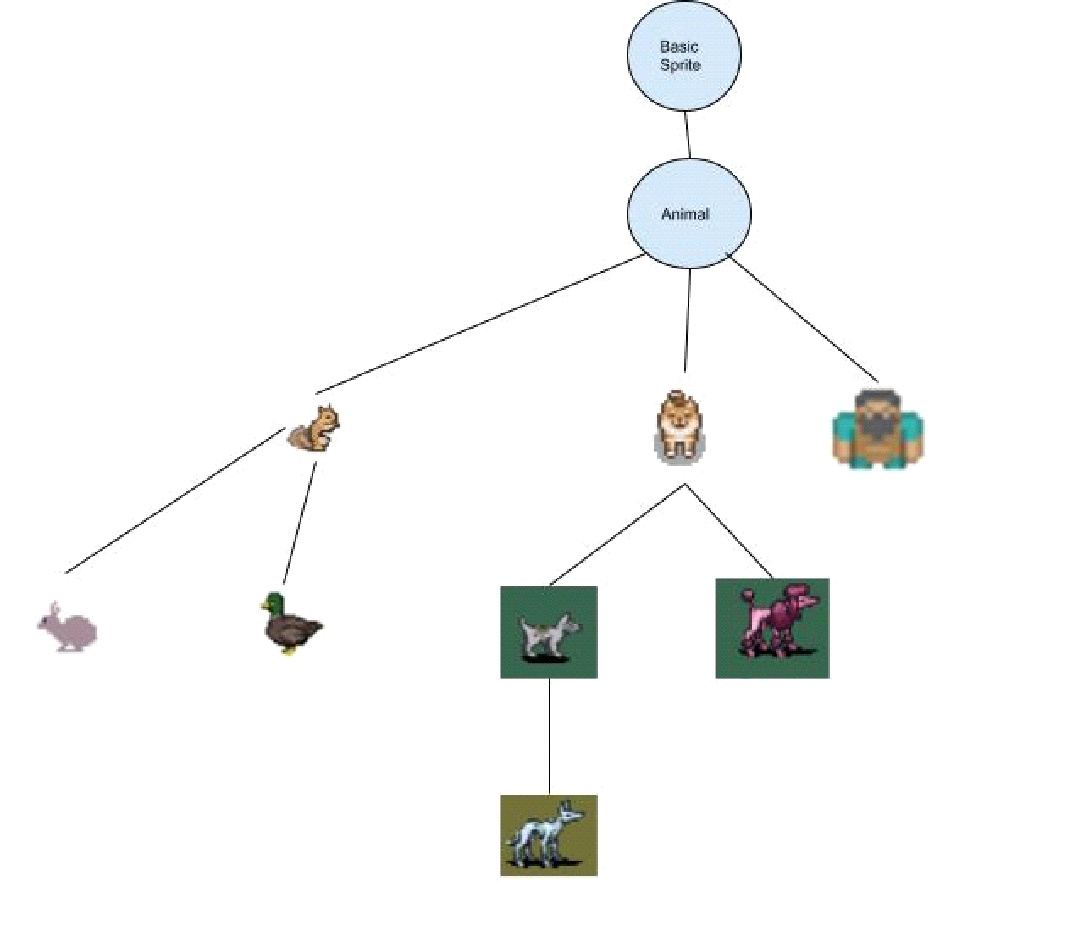
The animals terrain and items are stored in dictionary that stores the instances of each character type that is in the game.

{

"player":player,"dogs": [Dog1, Dog2, Dog3], "squirrels":[Squirrel1, Squirrel2], "trees":[Tree1, Tree2, Tree3], "items":[Ball]

}

**Object Orientated Programming**



The modification of an animal AI is efficiently achieved through the use of classes and inheritance.

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

eg. A new breed of dog can be created that inherits from the basic dog class and by modifying the function that controls the "return ball" state the new dog can be made to run away "flee" from the player once it has picked the ball, essentially making the dog play “chase me” with the player.

**Update Schedule**

**1st release 5th september**

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

**2nd update "Good Boy update" 10th September**

Features

-player can pat dog

-player can use the mouse to throw ball

-dog has basic animation.

-dog can poo

**3rd update "Clever boy update" 15th September**

Animals:

-Ducks

-Hares

Items

-Bread

Features"

-advanced pathfinding, weighted graph search

-diagonal movement

**4th update "Dog Day afternoon" 20th September**

Animals:

-Terrier -> Squirrel and duck Chasing

Features:

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

-player can have multiple dogs

-dogs can play together

**5th update "Dishlickers" 25th September**

Animals:

-human NPCs

-Grey hound

Features:

-grey hounds can be used for coursing hares

**6th Update "Barking Mad" 30th September**

Features

-Sounds

-NPC humans can have dogs

**7th Update "Angry Animals" 5th October**

Animals

-pitbull

-swan

Features

-Pitbull can attack 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 attack smaller dogs if player takes them too close.