Movement Pathfinding in Games

Subject: Artificial Intelligence

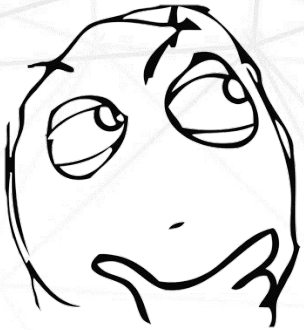
Professors: Edison jair Bejarano Sepulveda & Ramon Mateo Navarro

Course: 2024 / 2025

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

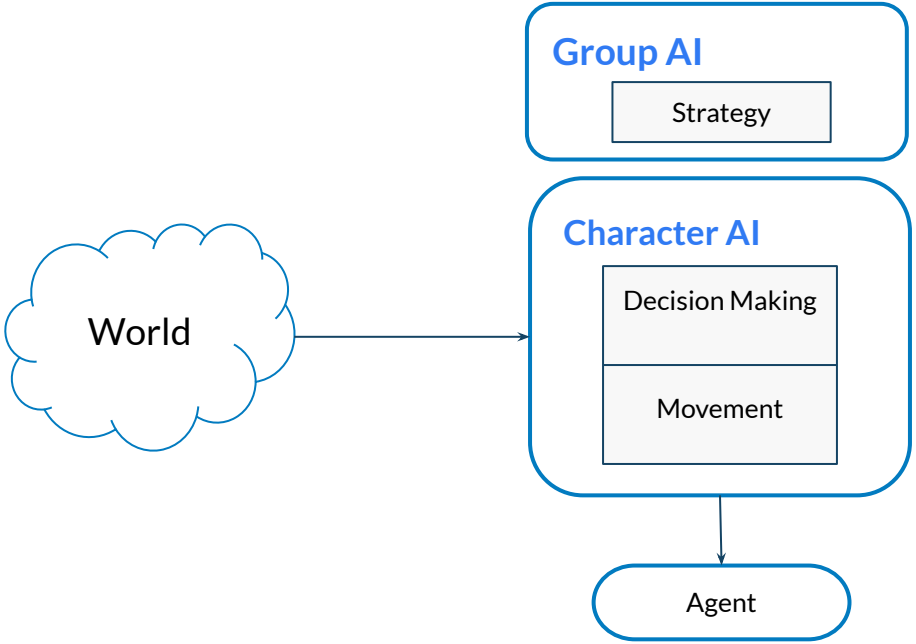
Overview

- **Introduction**
- NavMesh
- Steerings
- Flocking
- Graphs
- Pathfinding
- References



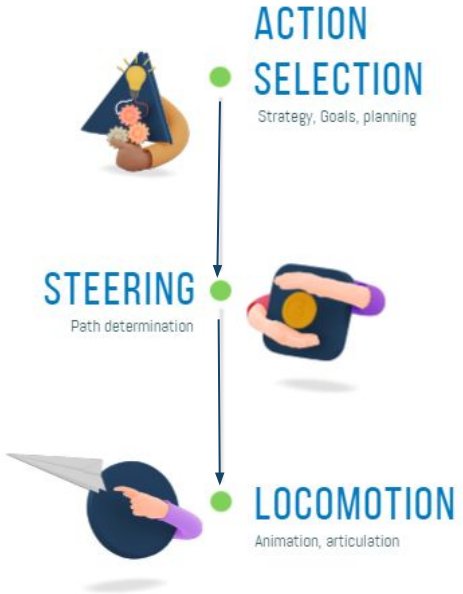
Movement

- Is the lowest level



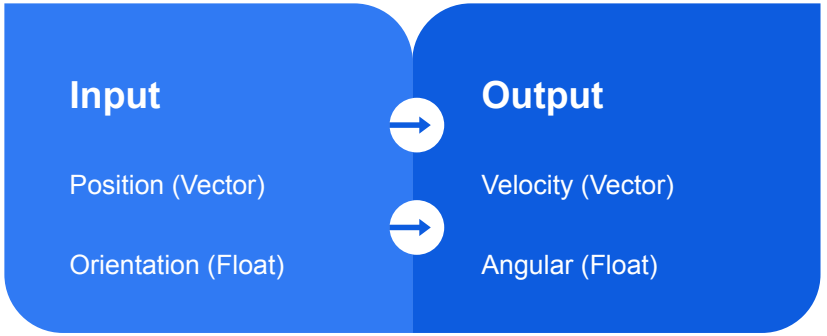
Hierarchy of movement behaviors

- **Steering**
  - Composed by simple atomic behaviors
  - They can be combined to behave very complex



Kinematic

- **Steering**
  - Simple behaviors (static)
  - Character as points(center of mass)
  - $2\frac{1}{2}D$: Hybrid 2D & 3D to simplify maths

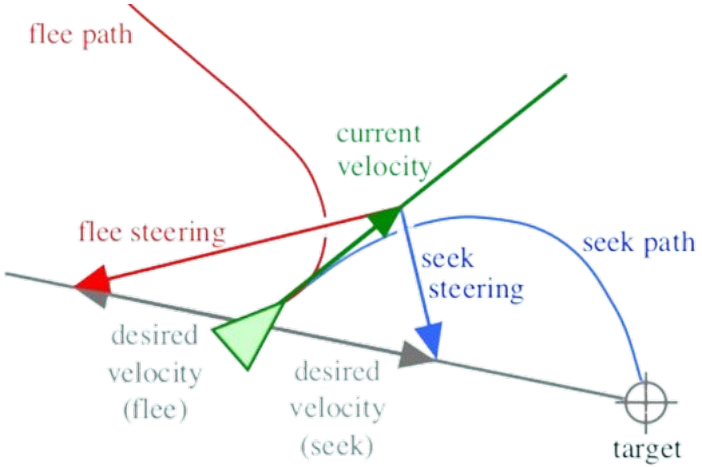


Position = Position + Velocity

Orientation = Orientation + Angle

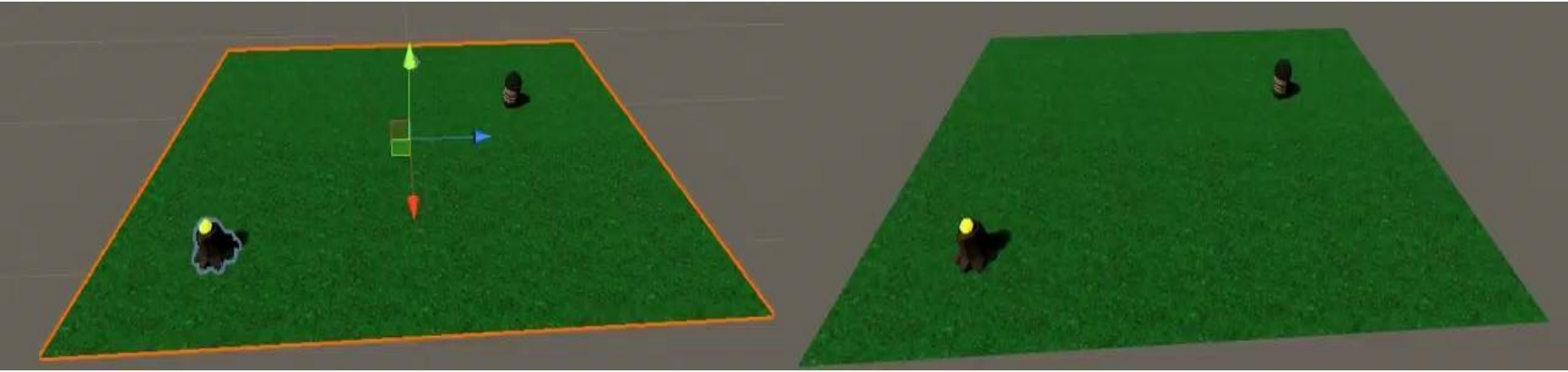
Kinematic Seek & Flee

- *It calculates the direction from the agent to the target.*
  - Input
    - Agent (Position, Orientation)
    - Target (Position)
    - max Velocity, max Rotation
  - Output
    - Velocity
    - Angle

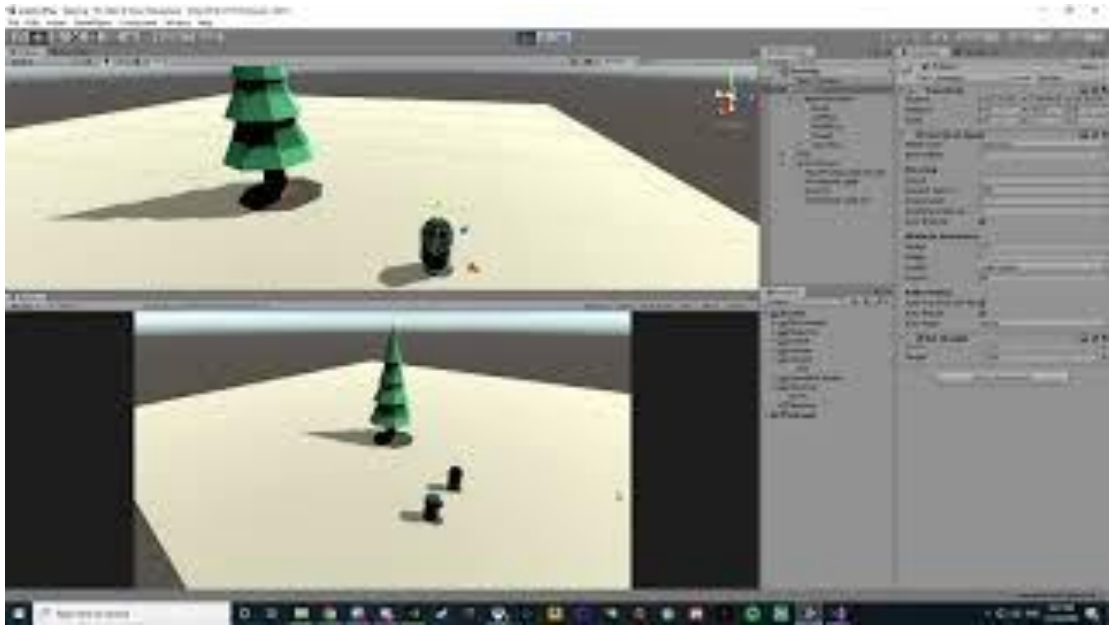


Source: (Reynolds, 1999)

Kinematic Seek & Flee (example)



Kinematic Seek & Flee (Examples)



Kinematic in Unity

direction: vector from robber to treasure

$$d = \left(t_x - r_x,\ 0,\ t_z - r_z \right)$$

```
// Seek
Vector3 direction = target.transform.position - transform.position;
direction.y = 0f;    // (x, z): position in the floor
// Flee
Vector3 direction = transform.position - target.transform.position;
```

velocity: vector direction with magnitude max Velocity

```
Vector3 movement = direction.normalized * maxVelocity;
```

rotation:

```
float angle = Mathf.Rad2Deg * Mathf.Atan2(movement.x, movement.z);
Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.up);  // up = y
```

Update and Time in Unity

Update: rotation and position (*dt = Time.deltaTime)*

```
transform.rotation = Quaternion.Slerp(transform.rotation, rotation,
                                      Time.deltaTime * turnSpeed);
transform.position += transform.forward.normalized * maxVelocity * Time.deltaTime;
```

Time: how to reduce frequency in steerings calls

```
float freq = 0f;
void Update()
{
    freq += Time.deltaTime;
    if (freq > 0.5)
    {
        freq -= 0.5f;
        Seek();
    }
    // Update commands
}
```

Math & Unity Stuff I

distance: between points

$$d(v, w) = \sqrt{\left(w_x - v_x\right)^2 + \left(w_z - v_z\right)^2}$$

```
Vector3.Distance(target.transform.position, transform.position)
```

Needed as **stoping criteria** to avoid wiggle in **seek**.

angle: between 2 vectors

```
Mathf.Abs(Vector3.Angle(transform.forward, movement)  // forward = z
```

dot product:

$$\langle v, w \rangle = v_x \cdot w_x + v_y \cdot w_y$$

$$\theta = \arccos \frac{\langle v, w \rangle}{|v||w|}$$

Math & Unity Stuff II

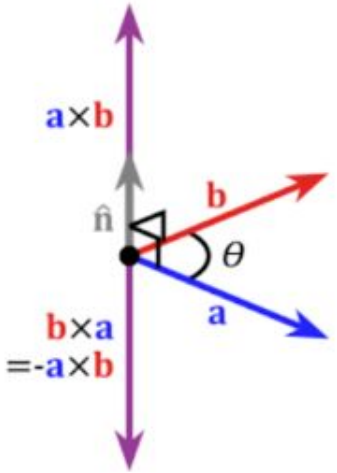
signed angle:

Vector3.SignedAngle(**v**, **w**, **transform**.forward)

base on cross product:

$$v \times w = (v_y \cdot w_z - v_z \cdot w_y, v_z \cdot w_x - v_x \cdot v_z, v_x \cdot w_y - v_y \cdot w_x)$$

- Clockwise: $(v \times w).z < 0$
- Anti-clockwise: $(v \times w).z < 0$



Steerings

- Kinematic drawback: it is not very realistic
- Steering (Dynamic) : by adding acceleration

Seek

Input:

- Agent (position, orientation)
- target (position)
- maxVelocity, maxRotation
- acceleration, turn Acceleration

Output:

- Velocity
- angle

Steering Update

```
void Update()
{
  if (Vector3.Distance(target.transform.position, transform.position) <
      stopDistance) return;
  Seek();   // calls to this function should be reduced
  turnSpeed += turnAcceleration * Time.deltaTime;
  turnSpeed = Mathf.Min(turnSpeed, maxTurnSpeed);
  movSpeed += acceleration * Time.deltaTime;
  movSpeed = Mathf.Min(movSpeed, maxSpeed);
  transform.rotation = Quaternion.Slerp(transform.rotation,
                                rotation, Time.deltaTime * turnSpeed);
  transform.position += transform.forward.normalized * movSpeed *
                    Time.deltaTime;
}
```

Steering Seek

```
void Seek()
{
   Vector3 direction = target.transform.position - transform.position;
   direction.y = 0f;
   movement = direction.normalized * acceleration;
   float angle = Mathf.Rad2Deg * Mathf.Atan2(movement.x, movement.z);
   rotation = Quaternion.AngleAxis(angle, Vector3.up);
}
```

Arriving

A chasing agent should never reach its goal when seeking

- Stopping distance
- Steering Arrive

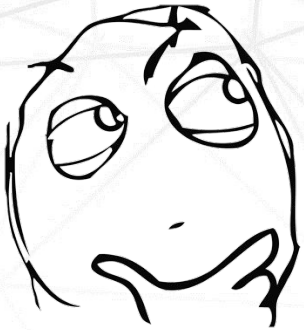
$$speed = \frac{maxSpeed}{times\ distance} - slowRadius$$

***Note**: maxAcceleration should be controlled*

Example:

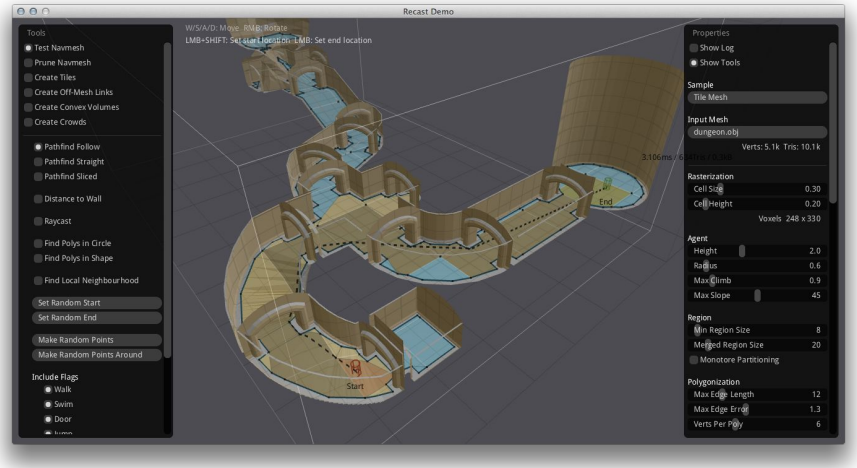
OUTLINE

- Introduction
- **NavMesh**
- Steerings
- Flocking
- Graphs
- Pathfinding
- References



Recast

- Library (Mononen, 2016) for pathfinding in 3d games
- Used by all major engines (state of the art)
- Also some proprietary engine (Horizon Zero Dawn)



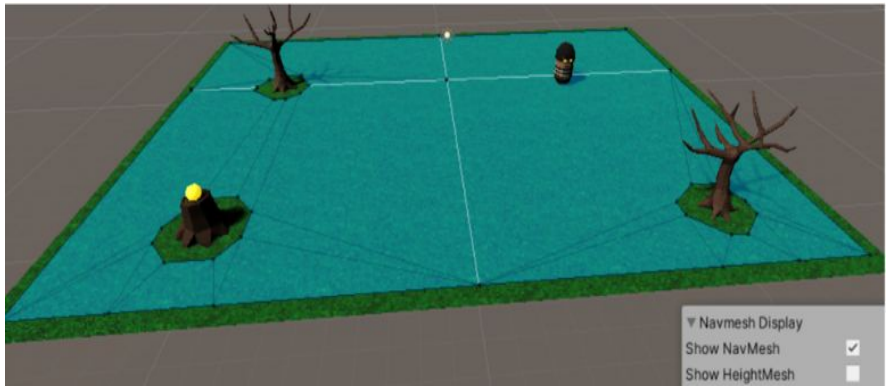
Navigation Mesh

NavMesh: polygon set representing walkable surfaces

NavMeshAgent: navigation component

OffMeshLink: navigation shortcuts

NavMeshObstacle: dynamic obstacle



https://docs.unity3d.com/Packages/com.unity.ai.navigation@1.1/manual/index.html

NavMesh

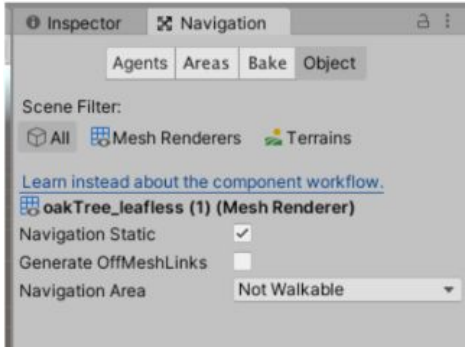
Creating the NavMesh:

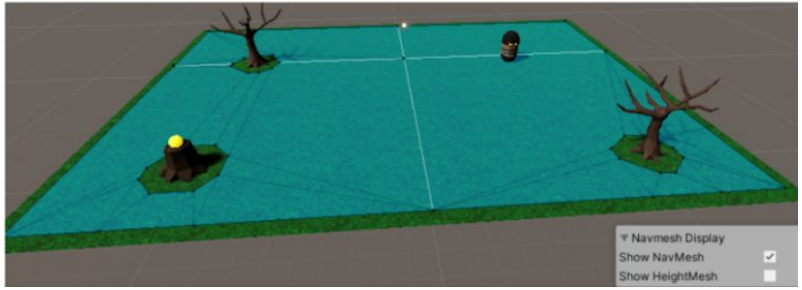
- Open `Window - AI - Navigation`
- Select scene objectes as:
  - `Static`
  - `Walkable` or `Not Walkable`
- Click `Bake` tab - `Bake` button

Bake again as you need

Main properties:

- Agent Radius & Height
- Max Slope & Step Height

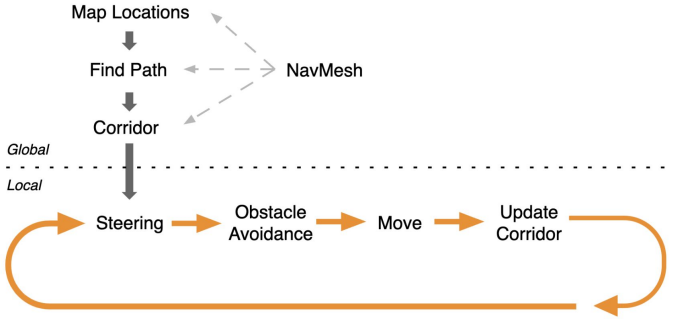


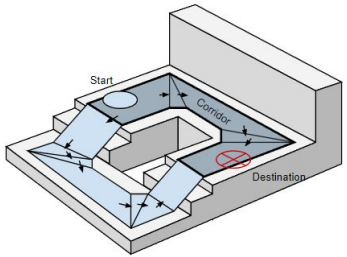


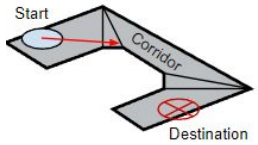
NavMesh Agent I

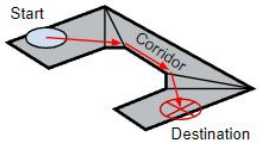
Inner Workings of the Navigation System:

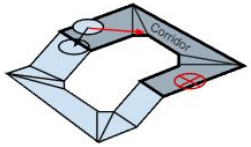
1. Find Paths
2. Follow the Path
3. Avoid Obstacles
4. Move the Agent (Steerings)

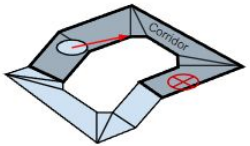












NavMesh Agent II

Using the NavMesh:

- Add the NavMesh Agent component to the agent
- Code

```
public NavMeshAgent agent;
public GameObject target;
void Seek()
{
    agent.destination = target.transform.position;
};
```

Main property groups:

- Steering Speed, Stopping Distance, Auto Braking…
- Object Avoidance: Radius…
- Path Finding: Auto Traverse off Mesh Links…

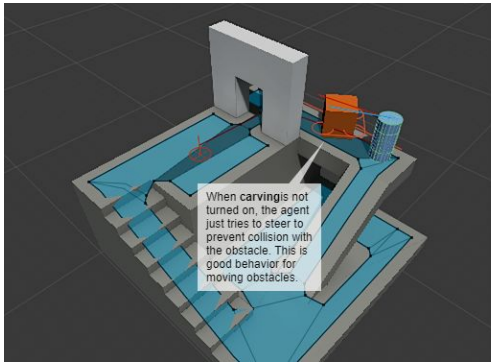
NavMesh Obstacle

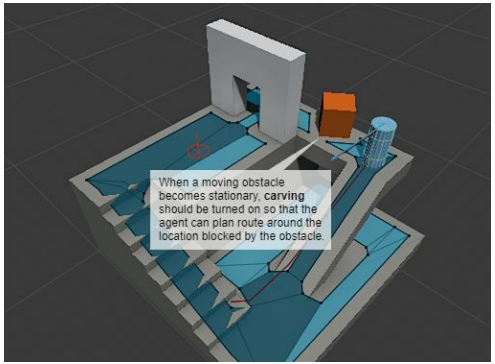
Creating a dynamic obstacle:

- Add the NavMesh Obstacle component to the object
- Add the RigidBody component to the object (being kinematic)

Main property(link documentation):

- Carve: creates a hole in the NavMesh

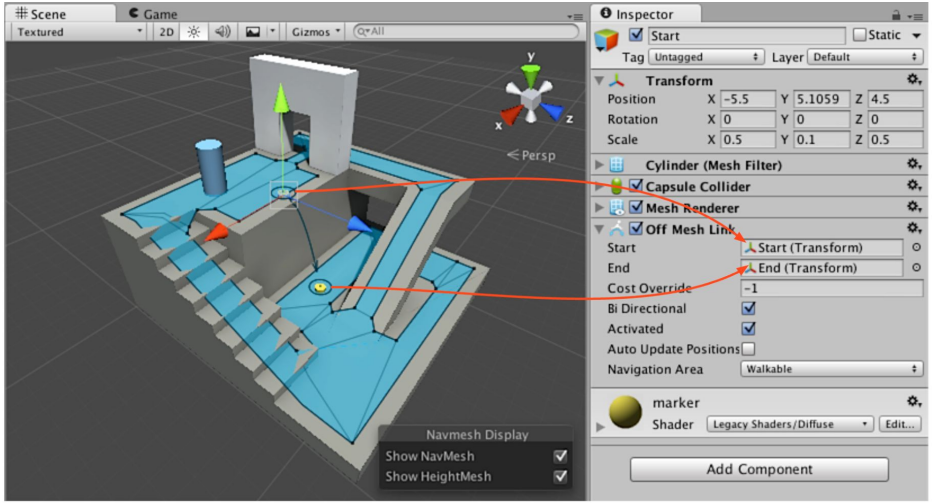




Off-Mesh Link I

Creating an off-mesh Link:

- Add the off Mesh Link component to one of the two objects



Source & documentation

Off-Mesh Link II

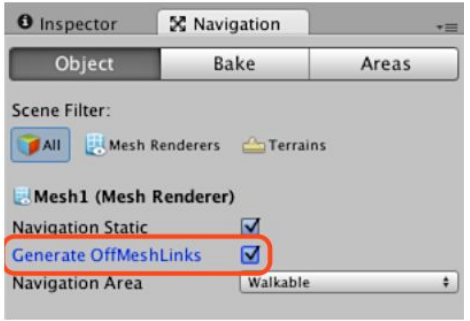
Building off-mesh Links

Tic the Generate OffMeshLinks at Navigation - Object

- Bake again

Main properties:

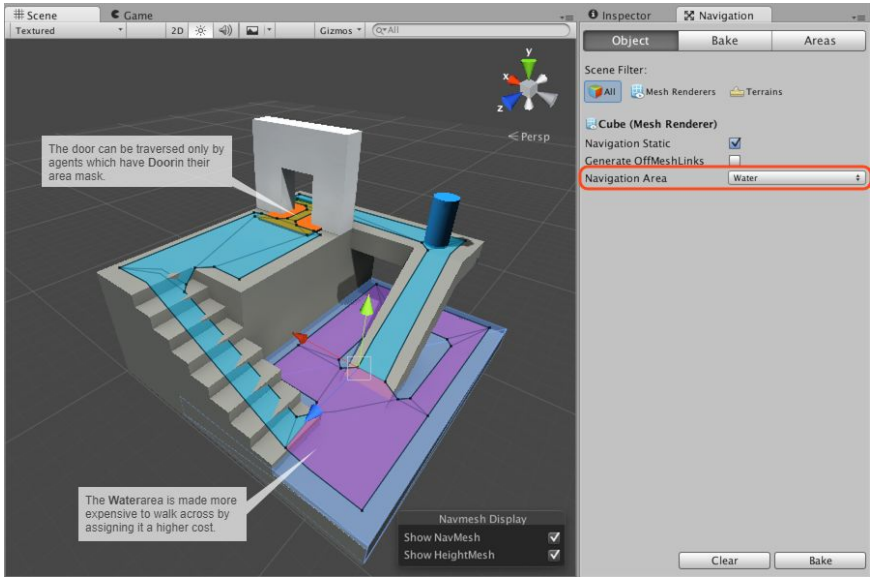
- Drop Height & Jump Distance



https://docs.unity3d.com/560/Documentation/Manual/nav-BuildingOffMeshLinksAutomatically.html

Navigation Areas and Costs

Navigation Areas define how difficult it is to walk across a specific area.



Source & documentation

Navigation System

A* search algorithm

- NavMeshAgent.SetDestination: possible not available at next frame
- NavMeshAgent.pathPending

NavMeshPath:

- Data structure: path as a list of waypoints
- NavMeshAgent.path: documentation

Advanced NavMesh

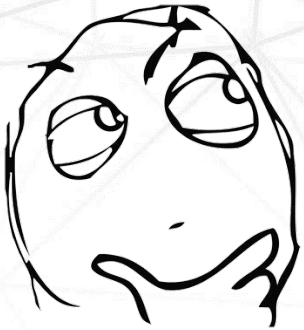
- Mesh Polygons
- NavMesh building components

Simple AI navigation tutorial

Link

Overview

- Introduction
- NavMesh
- **Steerings**
- Flocking
- Graphs
- Pathfinding
- References



Wander

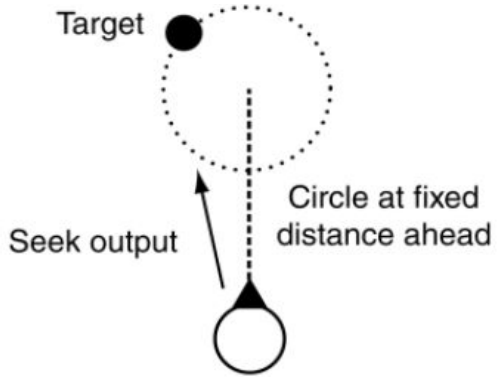
Simple implementation:

```
// parameters: float radius, offset;
Vector3 localTarget = UnityEngine.Random.insideUnitCircle * radius;
localTarget += new Vector3(0, 0, offset);
Vector3 worldTarget = transform.TransformPoint(localTarget);
worldTarget.y = 0f;
```

Issues:

- Remember *Auto Brake* & *Stopping Distance*
- How often calling wander?
- What about margins and objects?

NavMesh.SamplePosition



Wander

Example

Pursue & Evade

Simple implementation:

```
Vector3 targetDir = target.transform.position - transform.position;
float lookAhead = targetDir.magnitude / agent.speed;
Seek(target.transform.position + target.transform.forward * lookAhead);
// Flee for evasion
```

Examples:

pursuit

evasion

Hide (Previous C# Stuff)

Example

Defining Hiding Spots:

```
GameObject[] hidingSpots;
hidingSpots = GameObject.FindGameObjectsWithTag("hide");
```

Anonymous Functions:

```
Func<int, int> inc = (a) => a+ 1;
inc (4)) = 5
```

Tuples:

```
(int, string) a = (1, "Pep");
(int, string) b= (2, "Anna");

a.CompareTo(b) -1
```

Linq Select (Queries):

```
int[] v = {3, 2, -3, 5 };
v.Min()=-3
v.Select((x) => Math. Abs (x)).Min()
2
```

Hide

Simple implementation:

```
void Hide()
{
    Func<GameObject, float> distance =
        (hs) => Vector3.Distance(target.transform.position,
                                 hs.transform.position);
    GameObject hidingSpot = hidingSpots.Select(
        ho => (distance(ho), ho)
        ).Min().Item2;
    Vector3 dir = hidingSpot.transform.position - target.transform.position;
    Ray backRay = new Ray(hidingSpot.transform.position, -dir.normalized);
    RaycastHit info;
    hidingSpot.GetComponent<Collider>().Raycast(backRay, out info, 50f);
    Seek(info.point + dir.normalized);
}
```

Follow Path

```
public GameObject[] waypoints;
int patrolWP = 0;
...
if (!agent.pathPending && agent.remainingDistance < 0.5f) Patrol();
...
void Patrol()
{
  patrolWP = (patrolWP + 1) % waypoints.Length;
  Seek(waypoints[patrolWP].transform.position);
}
```

https://docs.unity3d.com/es/2018.4/Manual/nav-AgentPatrol.html

Smoothing the corners

Ghost Following:

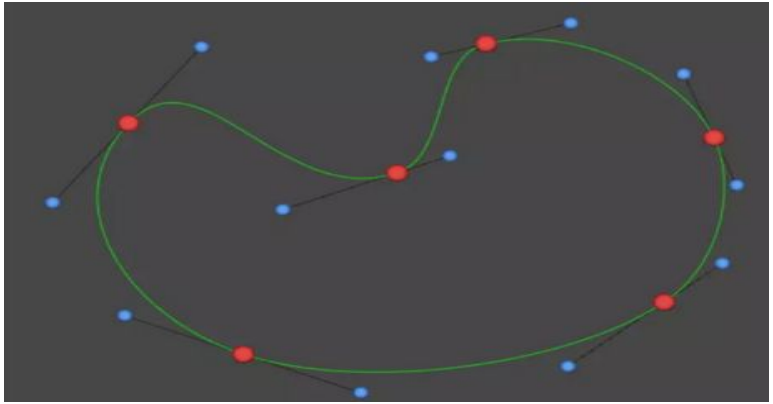
- Follow a ghost agent Example
- Adjust speeds (ghost waiting?)
- Remember to disable the ghost Mesh Renderer

Path Following with Beizer Curves:

- **BG Curve** asset. BansheeGz, 2020.
- **Bézier Path Creator** asset. Sebastian Lague, 2019.

  Example: video

Both contain getting closest point to the curve.



Smoothing the corners

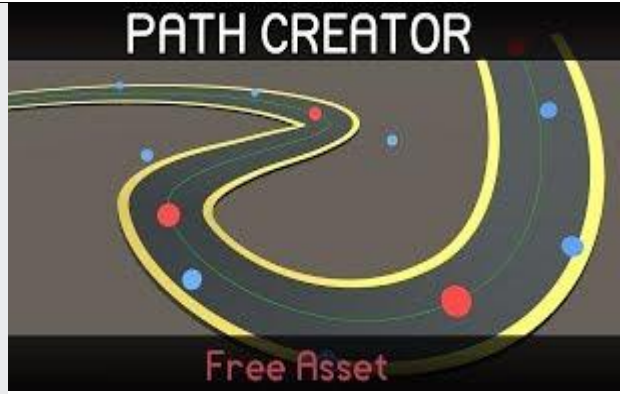
Example

1

```
using UnityEngine;
using UnityEngine.AI;
using PathCreation;
using System.Collections;

public class Follow : MonoBehaviour
{
    public GameObject robber;
    public GameObject treasure;
    public NavMeshAgent agent;
    public PathCreator pathCreator;
    public EndOfPathInstruction endOfPathInstruction;
    public float speed = 5;
    float distanceTravelled;

    void Start()
    {
        if (pathCreator != null)
        {
            distanceTravelled =
pathCreator.path.GetClosestDistanceAlongPath(transform.position);
            agent.destination =
pathCreator.path.GetPointAtDistance(distanceTravelled,
endOfPathInstruction);
        };
    }
```

2

```
    void Update()
    {
        if (Vector3.Distance(treasure.transform.position,
robber.transform.position) < 10f)
        {
            agent.destination = robber.transform.position;
            agent.isStopped = false;
        }
        else
            if (agent.remainingDistance > 0.2f)
            {
                distanceTravelled =
pathCreator.path.GetClosestDistanceAlongPath(transform.position);
                agent.destination =
pathCreator.path.GetPointAtDistance(distanceTravelled,
endOfPathInstruction);
            }
            else
            {
                agent.isStopped = true;
                if (pathCreator != null)
                {
                    distanceTravelled += speed * Time.deltaTime;
                    transform.position =
pathCreator.path.GetPointAtDistance(distanceTravelled,
endOfPathInstruction);
                    transform.rotation =
pathCreator.path.GetRotationAtDistance(distanceTravelled,
endOfPathInstruction);
                }
            }
    }
}
```



- Bézier Path Creator asset. Sebastian Lague, 2019.
  - Example:

    video/ Code

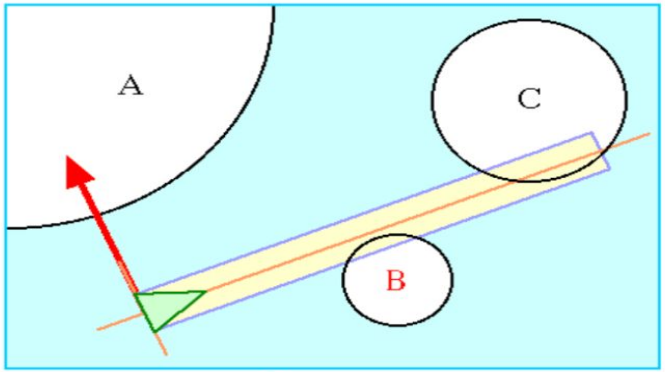
Combining Steering Behaviors

- Previours steerings serve as building blocks for complex behaviors.
- Combination can happen in many ways:
  - Arbitration: switch steerings as world changes Example: wander & pursue
  - Blending: sum or weighted sum Example: flocking (*separation* + *align* + *cohesion*) Problem: **components cancelling**
  - Mixing arbitration and blending
- Advanced combinations:
  - Priority groups: blending plus priorities execute highest priority steerings and ignore the rest
  - More complex structures: Cooperative Arbitration

Combinations need to be carefully adjusted.

Steering Stuff

- There are many more movements (see references): Example: Obstacle and Wall Avoidance



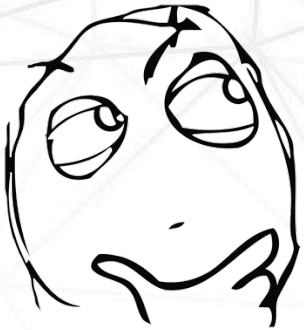
https://www.red3d.com/cwr/steer/gdc99/

- Reynolds OpenSteer

  C++ library to help construct steering behaviors for autonomous characters in games and animation

Overview

- Introduction
- NavMesh
- Steerings
- **Flocking**
- Graphs
- Pathfinding
- References

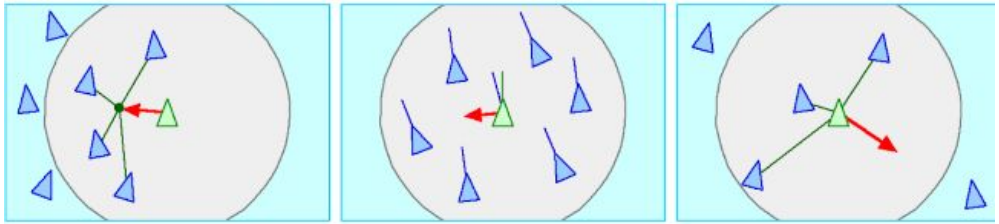


Flocking

Groupal behavior such of birds or fishes

Sum of three simple rules:

- Cohesion: neighbour center of mass
- Match velocity/align: average neighbours heading
- Separation: avoid crowding neighbours

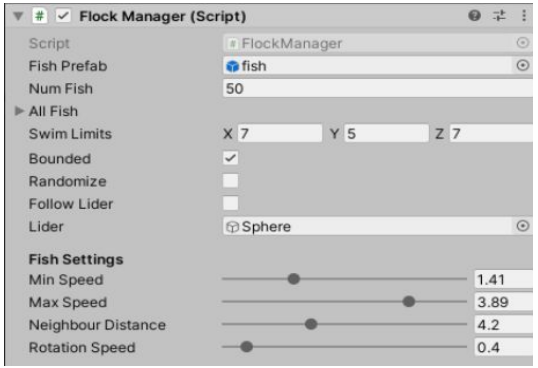


https://www.youtube.com/watch?v=mjKINQigAE4&list=PL5KbKbJ6Gf99UlyIqzV1UpOzseyRn5H1d

FLOCKING SETTINGS

Flocking Manager

```
allFish = new GameObject[numFish];
for (int i = 0; i < numFish; ++i) {
   Vector3 pos = this.transform.position + ... // random position
   Vector3 randomize = ... // random vector direction
   allFish[i] = (GameObject)Instantiate(fishPrefab, pos,
                           Quaternion.LookRotation(randomize));
   allFish[i].GetComponent<Flock>().myManager = this;
}
```



Flocking Rules I

Cohesion:

```
Vector3 cohesion = Vector3.zero;
int num = 0;
foreach (GameObject go in myManager.allFish) {
   if (go != this.gameObject) {
      float distance = Vector3.Distance(go.transform.position,
                                        transform.position);
      if (distance <= myManager.neighbourDistance) {
         cohesion += go.transform.position;
         num++;
      }
   }
}
if (num > 0)
   cohesion = (cohesion / num - transform.position).normalized * speed;
```

Flocking Rules II

Match velocity/aling:

```
Vector3 align = Vector3.zero;
int num = 0;
foreach (GameObject go in myManager.allFish) {
   if (go != this.gameObject) {
      float distance = Vector3.Distance(go.transform.position,
                                        transform.position);
      if (distance <= myManager.neighbourDistance) {
         align += go.GetComponent<Flock>().direction;
         num++;
      }
   }
}
if (num > 0) {
   align /= num;
   speed = Mathf.Clamp(align.magnitude, myManager.minSpeed, myManager.maxSpeed);
}
```

Flocking Rules III

Separation:

```
Vector3 separation = Vector3.zero;
foreach (GameObject go in myManager.allFish) {
  if (go != this.gameObject) {
    float distance = Vector3.Distance(go.transform.position,
                                      transform.position);
    if (distance <= myManager.neighbourDistance)
      separation -= (transform.position - go.transform.position) /
                    (distance * distance);
  }
}
```

More Flocking Stuff

Combination:

```
direction = (cohesion + align + separation).normalized * speed;
```

- Three rules + combination should be placed in the same foreach.

Update:

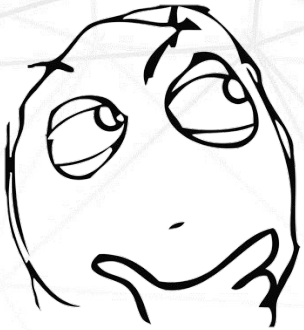
```
transform.rotation = Quaternion. Slerp(transform.rotation,
                                       Quaternion.LookRotation(direction),
                                       myManager.rotationSpeed* Time.deltaTime);
transform. Translate(0.0f, 0.0f, Time.deltaTime speed);
```

Final notes:

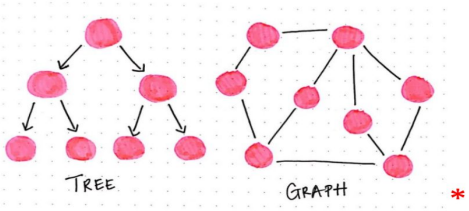
- Rules should not be calculated every frame.
- Some random issues enriches the behaviour.
- Introduction of a lider is a common extension.

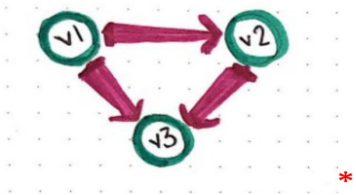
Overview

- Introduction
- NavMesh
- Steerings
- Flocking
- **Graphs**
- Pathfinding
- References



GRAPHS





Math definition:

$G = (V, E)$

V = set of vertices

E = set of edges

Example:

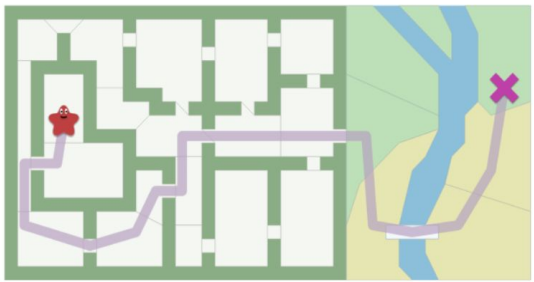
$V = \{v_1, v_2, v_3\}$

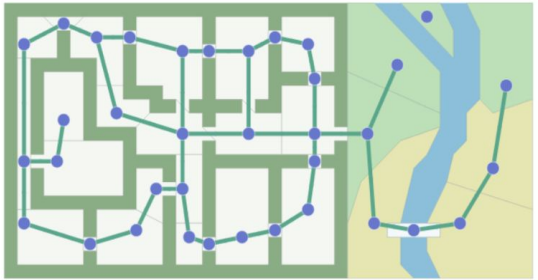
$E = \{(v_1, v_2), (v_1, v_3), (v_2, v_3)\}$

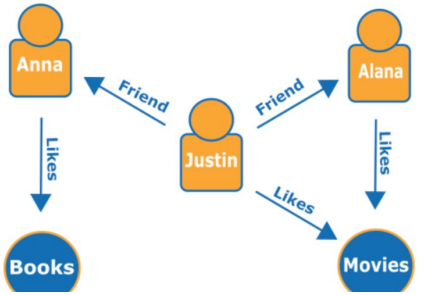
- Edges can be directed (one way) or undirected (two ways).
- Both vertices and edges can contain information.

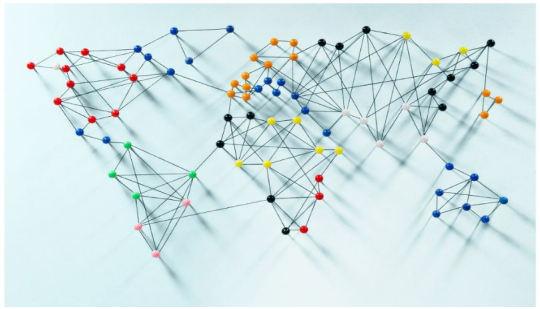
https://medium.com/basecs/a-gentle-introduction-to-graph-theory-77969829ead8

REPRESENTATION AS GRAPHS



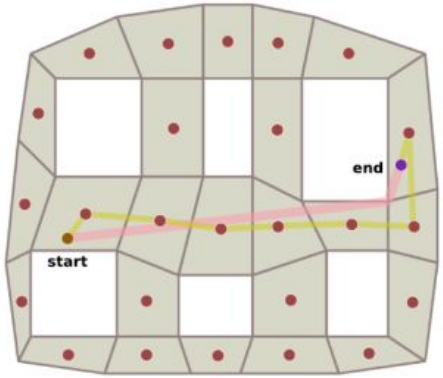






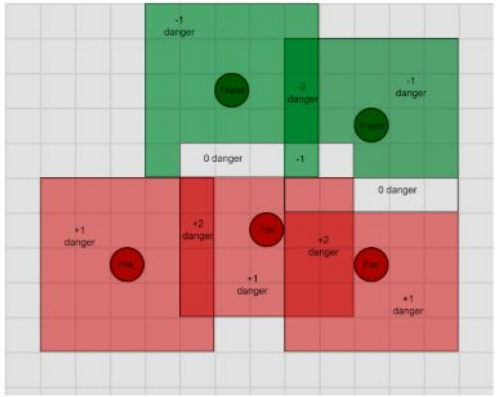
Some Applications in GameAI

Pathfinding:



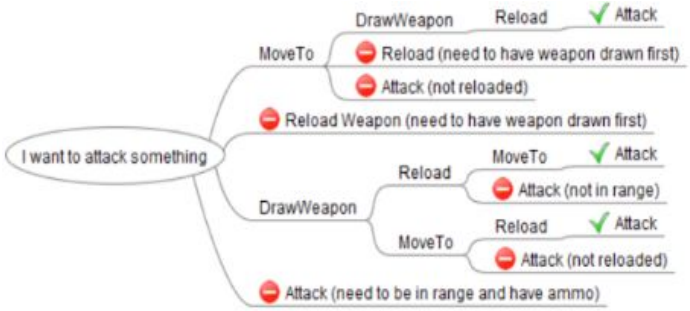
https://www.indiedb.com/games/attack-of-the-gelatinous-blob/news/a-look-into-the-ai-goap

Tactics: influence maps



Pathfinding source

Decision making: planners



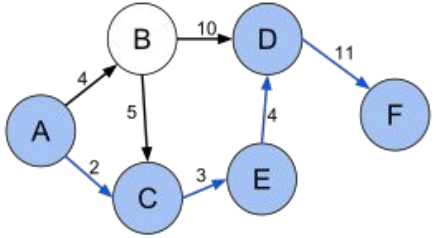
https://www.gamedev.net/articles/programming/artificial-intelligence/the-total-beginners-guide-to-game-ai-r4942/

Shortest Path Problem

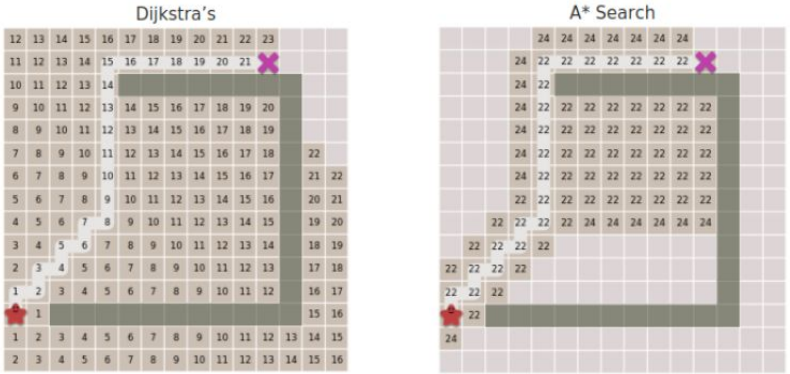
Find the minimum (sum of edges costs) path between two vertices. Main Algorithms:

Dijkstra: general cases

A:requires an heuristic h (estimation cost function)



https://en.wikipedia.org/wiki/Shortest_path_problem



https://www.redblobgames.com/pathfinding/a-star/introduction.html

Dijkstra

Pseudocode:

Pseudocode:

$Q \leftarrow V(G)$
$d \leftarrow [\infty, \forall v \in V(G)]$
$d[source] \leftarrow 0$
while not empty(Q) **do**
  $v \leftarrow \text{argmin}_x \min_{\forall x \in Q}(d(x))$
  Q.remove(v)
  for all $u \in$ neighbours(v) **do**
    if $d(v) + \text{edge}(v, u) \leq d(u)$ **then**
      $d(u) \leftarrow d(v) + \text{edge}(v, u)$
    end if
  end for
end while

Source

https://courses.cs.duke.edu//fall11/cps149s/notes/a_star.pdf

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

1

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;
using System;

public class Graph
{
    public HashSet<string> vertices = new HashSet<string> {"a", "b", "c", "d", "e", "f"};

    public Dictionary<string,Dictionary<string,int>> edges = new
Dictionary<string,Dictionary<string,int>>
        {
            { "a", new Dictionary<string,int> { {"b", 4}, {"c", 2} } },
            { "b", new Dictionary<string,int> { {"c", 5}, {"d", 10} } },
            { "c", new Dictionary<string,int> { {"e", 3} } },
            { "d", new Dictionary<string,int> { {"f", 11} } },
            { "e", new Dictionary<string,int> { {"d", 4} } },
            { "f", new Dictionary<string,int> { } }
        };

    public Dictionary<string, int> h = new Dictionary<string, int>
        { {"a", 20}, {"b", 18}, {"c", 12}, {"d", 10}, {"e", 9}, {"f", 0} };
}

public class shortestPath : MonoBehaviour
{
    void Start()
    {
        Graph g = new Graph();
        Show(Dijkstra(g, "a", "f"));
```

2

```
public class shortestPath : MonoBehaviour
{
    void Start()
    {
        Graph g = new Graph();
        Show(Dijkstra(g, "a", "f"));
        Show(Astar(g, "a", "f"));
    }

    List<string> Dijkstra(Graph g, string source, string target)
    {
        var d = g.vertices.ToDictionary(v => v, v => 1000);
        var prev = g.vertices.ToDictionary(v => v, v => " ");
        d[source] = 0;
        HashSet<String> Q = new HashSet<string>(g.vertices);
        while (Q.Count() > 0)
        {
            string v = Q.Select(x => (d[x], x)).Min().Item2;
            Q.Remove(v);
            foreach (var pair in g.edges[v])
            {
                int alt = d[v] + pair.Value;
                if (alt < d[pair.Key])
                {
                    d[pair.Key] = alt;
                    prev[pair.Key] = v;
                }
            }
        }
        List<string> path = new List<string>();
        path.Insert(0,target);
        while (prev[target] != " ")
        {
            target = prev[target];
            path.Insert(0,target);
        }
        return path;
    {
            target = prev[target];
            path.Insert(0,target);
        }
```

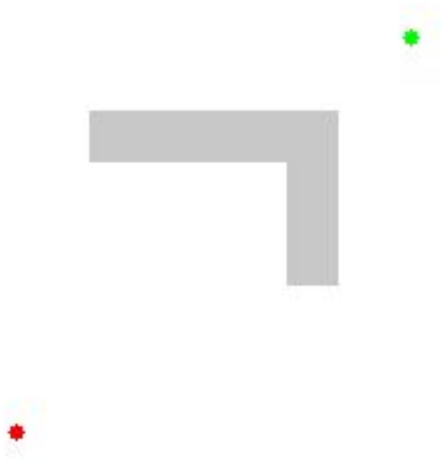
3

```
        return path;
    }

    List<string> Astar(Graph g, string source, string target)
    {
        var d = g.vertices.ToDictionary(v => v, v => 1000);
        var prev = g.vertices.ToDictionary(v => v, v => " ");
        d[source] = 0;
        HashSet<String> Q = new HashSet<string>(g.vertices);
        while (Q.Count() > 0)
        {
            string v = Q.Select(x => (d[x] + g.h[x], x)).Min().Item2;
            Q.Remove(v);
            foreach (var pair in g.edges[v])
            {
                int alt = d[v] + pair.Value;
                if (alt < d[pair.Key])
                {
                    d[pair.Key] = alt;
                    prev[pair.Key] = v;
                }
            }
        }

        List<string> path = new List<string>();
        path.Insert(0,target);
        while (prev[target] != " ")
        {
            target = prev[target];
            path.Insert(0,target);
        }
        return path;
    }

    void Show(List<string> l)
    {
        string s = "Path:\n";
        foreach (var x in l)
            s += " " + x;
        Debug.Log(s);
    }
}
```

A*

Pseudocode:

$Q \leftarrow V(G)$
$d \leftarrow [\infty, \forall v \in V(G)]$
$d[source] \leftarrow 0$
while not empty(Q) **do**
  $v \leftarrow \text{argmin}_x \min_{\forall x \in Q}(d(x) + h(x))$
  Q.remove(v)
  for all $u \in$ neighbours(v) **do**
    if $d(v) + \text{edge}(v, u) \leq d(u)$ **then**
      $d(u) \leftarrow d(v) + \text{edge}(v, u)$
    end if
  end for
end while



https://courses.cs.duke.edu//fall11/cps149s/notes/a_star.pdf

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

1

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;
using System;

public class Graph
{
    public HashSet<string> vertices = new HashSet<string> {"a", "b",
"c", "d", "e", "f"};

    public Dictionary<string,Dictionary<string,int>> edges = new
Dictionary<string,Dictionary<string,int>>
        {
            { "a", new Dictionary<string,int> { {"b", 4}, {"c", 2} } },
            { "b", new Dictionary<string,int> { {"c", 5}, {"d", 10} } },
            { "c", new Dictionary<string,int> { {"e", 3} } },
            { "d", new Dictionary<string,int> { {"f", 11} } },
            { "e", new Dictionary<string,int> { {"d", 4} } },
            { "f", new Dictionary<string,int> { } }
        };

    public Dictionary<string, int> h = new Dictionary<string, int>
        { {"a", 20}, {"b", 18}, {"c", 12}, {"d", 10}, {"e", 9}, {"f", 0} };
}

public class shortestPath : MonoBehaviour
{
    void Start()
    {
        Graph g = new Graph();
        Show(Dijkstra(g, "a", "f"));
        Show(Astar(g, "a", "f"));
    }

    List<string> Dijkstra(Graph g, string source, string target)
```

2

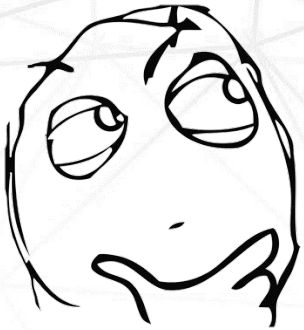
```
 {
        var d = g.vertices.ToDictionary(v => v, v => 1000);
        var prev = g.vertices.ToDictionary(v => v, v => " ");
        d[source] = 0;
        HashSet<String> Q = new
HashSet<string>(g.vertices);
        while (Q.Count() > 0)
        {
            string v = Q.Select(x => (d[x], x)).Min().Item2;
            Q.Remove(v);
            foreach (var pair in g.edges[v])
            {
                int alt = d[v] + pair.Value;
                if (alt < d[pair.Key])
                {
                    d[pair.Key] = alt;
                    prev[pair.Key] = v;
                }
            }
        }
        List<string> path = new List<string>();
        path.Insert(0,target);
        while (prev[target] != " ")
        {
            target = prev[target];
            path.Insert(0,target);
        }
        return path;
    }
```

3

```
List<string> Astar(Graph g, string source, string target)
    {
        var d = g.vertices.ToDictionary(v => v, v => 1000);
        var prev = g.vertices.ToDictionary(v => v, v => " ");
        d[source] = 0;
        HashSet<String> Q = new HashSet<string>(g.vertices);
        while (Q.Count() > 0)
        {
            string v = Q.Select(x => (d[x] + g.h[x], x)).Min().Item2;
            Q.Remove(v);
            foreach (var pair in g.edges[v])
            {
                int alt = d[v] + pair.Value;
                if (alt < d[pair.Key])
                {
                    d[pair.Key] = alt;
                    prev[pair.Key] = v;
                }
            }
        }

        List<string> path = new List<string>();
        path.Insert(0,target);
        while (prev[target] != " ")
        {
            target = prev[target];
            path.Insert(0,target);
        }
        return path;
    }

    void Show(List<string> l)
    {
        string s = "Path:\n";
        foreach (var x in l)
            s += " " + x;
        Debug.Log(s);
    }
}
```

Overview

- Introduction
- NavMesh
- Steerings
- Flocking
- Graphs
- **Pathfinding**
- References



Pathfinding

Components:

- World Representation as graphs
  - Vertices: convex surfaces

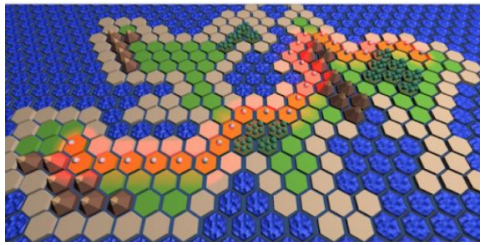
    no line segment between two inner points goes outside the surface
  - Edges: connect vertices with cost
- A* algorithm:

choosing a heuristic

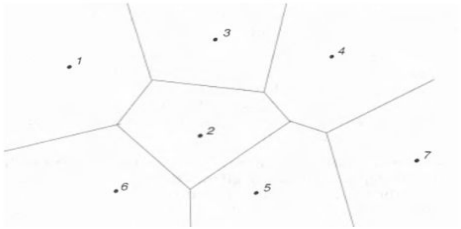
- Path Smoothing
  - algorithm

World Representation

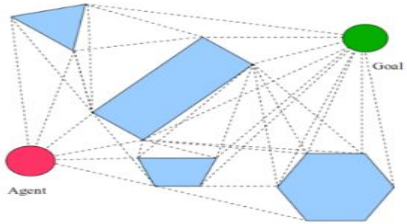
Tile Graphs: world splitted in regular tiles (squares, hexagons…)



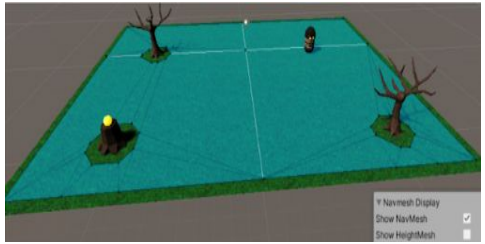
Dirichlet Domains: regions defined (manually) by a set points



Points of Visibility:



Navigation Meshes



Heuristics

Properties:

- **Underestimating**: heuristic too slow. The more accurate the faster A* runs.
- **Overestimating**: heuristic too high.

  A* my not return the best path.
- **Admissible**: if an heuristic h(n) is lower than the true cost for all the nodes, A* is optimal.

Some common heuristics:

- **Euclidean distance**

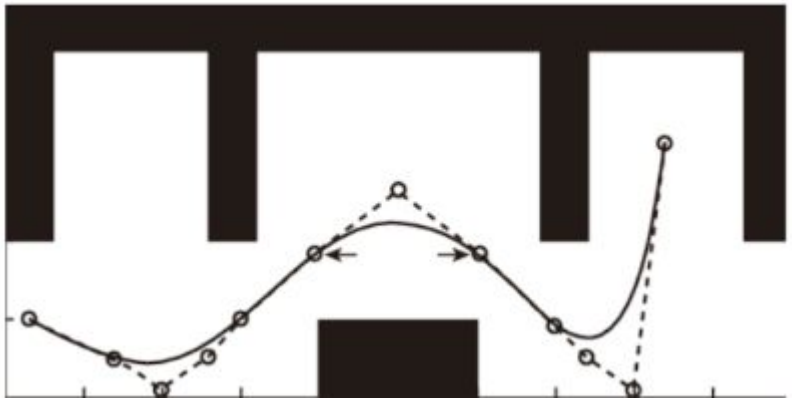
  In presence of lot of walls and corridor (indoor levels) it takes longer to run.
- **Cluster Heuristic**: grouping graph vertices together in clusters. Every room becomes a cluster. Automatic or provided by de level designer.

Path Smoothing

Points of Visibility:

$Q \leftarrow V(G)$
$d \leftarrow [\infty, \forall v \in V(G)]$
$d[source] \leftarrow 0$
while not empty(Q) **do**
  $v \leftarrow \text{argmin}_x \min_{\forall x \in Q}(d(x))$
  Q.remove(v)
  for all $u \in$ neighbours(v) **do**
    if $d(v) + \text{edge}(v, u) \leq d(u)$ **then**
      $d(u) \leftarrow d(v) + \text{edge}(v, u)$
    end if
  end for
end while

Points of Visibility:

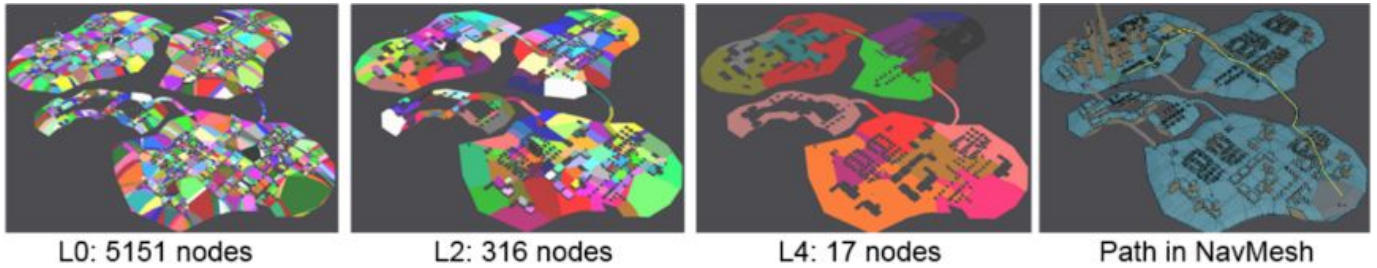


https://www.researchgate.net/figure/Path-smoothing-with-Bezier-curve_fig4_285739464

Hierarchical Pathfinding

Main idea:

- Clustering: group nodes to build a higher level graph
- Connection costs:
  minimum, maximum or average distances
- Pathfinding:
  - Apply pathfinding on higher level graph
  - For each cluster in resulting path apply pathfinding



Hierarchical Path-Finding for Navigation Meshes https://www.cs.upc.edu/~npelechano/Pelechano_HNAstar_prePrint.pdf

Other A* Variations

- Open Goal Pathfinding: many possible goals.

  Example: alarms

- Dynamic Pathfinding (D*): changing evironment (allows backtracking)

  Example: change the route to avoid detection

- Low Memory Algorithms:
  - IDA*: no lists
  - SMA*: fixed size open list

- Pooling Planners: queue of pathfinders.

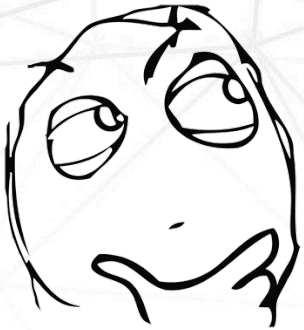
  Example: MMORG

- Continuous Time Pathfinding: task changes quickly (JPS+)

  Example: Racing Games

Overview

- Introduction
- NavMesh
- Steerings
- Flocking
- Graphs
- Pathfinding
- **References**



References

- Ian Millington. *AI for Games* (3rd ed). CRC Press, 2019.
- Craig W. Reynolds. **Steering Behaviors For autonomous Characters**. Proceedings of the Game Developers Conference (GDC), 1999.
- Penny de Byl. **Artificial Intelligence for Beginners**. Unity Learn Course, 2020.
- Sebastian Lague. **Boids (Flocking, github). Video**, 2019.

Libraries

- Craig W. Reynolds. **OpenSteer**, 2004.
- Mikko Mononen. **Recast & Detour**, 2016.

Resources

Examples:

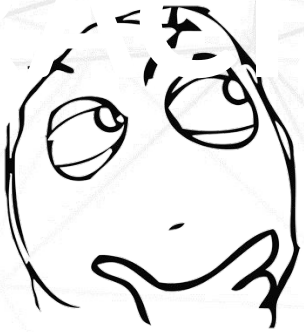
- Easy Primitive People asset. Bit Gamey, 2020.
- LowPoly Trees and Rocks asset. greyRoad Studio, 2019.
- Five Seamless Tileable Ground Textures asset. A3D, 2020.
- Simplistic Low Poly Nature asset. Acorn Bringer, 2018.

Bezier Curves:

- BG Curve asset. BansheeGz, 2020.
- Bézier Path Creator asset. Sebastian Lague, 2019.

Image

- Fondo Marino. Alejandro Muñoz Cabrisas, 2017.



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia