Navigation

IML

Cédric Buche

ENIB

30 août 2018

Page 1 :

Classical approaches
Video games
GNG
SGNG

Goal

Navigation

- determine paths to go from one point to an other.

Page 2 :

1. Classical approaches
   - Mesh
   - Graph
2. Video games
   - Quake 3 Arena bots
   - Counter-Strike bots
   - Unreal Tournament's series bots
   - Valve bots
3. GNG
4. SGNG

IML

2018-08-30

1. Classical approaches
   - Mesh
   - Graph
2. Video games
   - Quake 3 Arena bots
   - Counter-Strike bots
   - Unreal Tournament's series bots
   - Valve bots
3. GNG
4. SGNG

Page 3 :

1. Classical approaches
   - Mesh
   - Graph
2. Video games
   - Quake 3 Arena bots
   - Counter-Strike bots
   - Unreal Tournament's series bots
   - Valve bots
3. GNG
4. SGNG

IML
└─Classical approaches

2018-08-30

1. Classical approaches
   - Mesh
   - Graph
2. Video games
   - Quake 3 Arena bots
   - Counter-Strike bots
   - Unreal Tournament's series bots
   - Valve bots
3. GNG
4. SGNG

Page 4 :

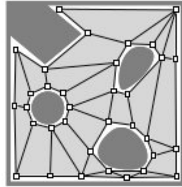
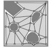


FIGURE – A simple environment (obstacles are in dark grey) represented by a mesh. The avatar can navigate in the zone defined by the mesh (in grey) because it knows there are no obstacles in this zone. Different algorithms can be used to find optimal paths.

Problems

- requires an algorithm to find the optimal path between two points
- a path which may not be natural or believable
- All the meshes used to render the environment are too complex for agents to handle.



Page 5 :

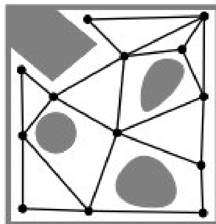
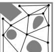


FIGURE – A simple environment (obstacles are in grey) represented by a graph. Nodes are represented by circles and edges by black lines. An avatar can move from one node to another only if the nodes are connected by an edge. Usually, an A* is used to find the path between two nodes.

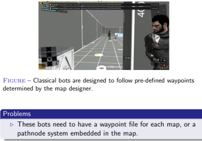


Page 6 :

FIGURE – Classical bots are designed to follow pre-defined waypoints determined by the map designer.

Problems

- ▷ These bots need to have a waypoint file for each map, or a pathnode system embedded in the map.



Page 7 :

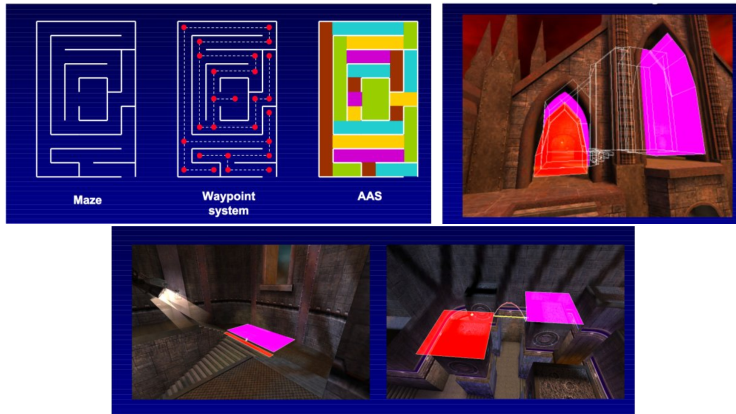
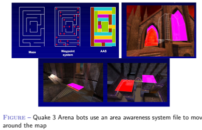


FIGURE – Quake 3 Arena bots use an area awareness system file to move around the map



Page 8 :

- ▷ The structure of the levels is analyzed, which is used to extract volumes of space that are navigable (e.g. walkable surfaces, swimming areas).
- ▷ A process then analyzes these areas for connections by simulating potential movement from each of them, and then clusters the areas together to allow hierarchical path-finding
- ▷ Quake 3 uses a form of pathfinding within the AAS graph that relies on the clusters of areas.
- ▷ Within these clusters, the paths to common destinations are pre-calculated and stored for later use.

▷ The structure of the levels is analyzed, which is used to extract volumes of space that are navigable (e.g. walkable surfaces, swimming areas).
▷ A process then analyzes these areas for connections by simulating potential movement from each of them, and then clusters the areas together to allow hierarchical path-finding
▷ Quake 3 uses a form of pathfinding within the AAS graph that relies on the clusters of areas.
▷ Within these clusters, the paths to common destinations are pre-calculated and stored for later use.

Page 9 :

Counter-Strike bots use a waypoint file.

Counter-Strike bots use a waypoint file.

Page 10 :

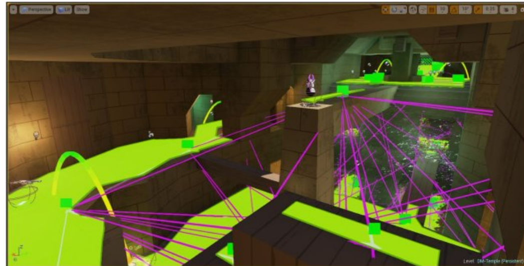


Figure – Unreal Tournament's series bots use a pathnode system embedded in the map to navigate

2018-08-30

IML
└Video games
 └Unreal Tournament's series bots

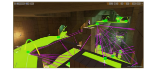


Figure – Unreal Tournament's series bots use a pathnode system embedded in the map to navigate

Page 11 :

- ▷ To support the many community-created maps, some games include an automatic mesh generation system
- ▷ The first time users attempt to play a custom map with bots, the generation system will build a navigation file for that map. Starting at a player spawn point, walkable space is sampled by "flood-filling" outwards from that spot, searching for adjacent walkable points.
- ▷ Finally, dynamic bots are able to dynamically learn levels and maps as they play. RealBot, for Counter-Strike, is an example of this. However, this learning is not guided by human behavior.
- ▷ Navigation points obtained will therefore not produce believable behavior. The paths the bots use to go from one point in the environment to another do not resemble those human players would take. This problem comes not from the decision-making process itself, but from the representation of the environment it uses. Indeed, the bots use navigation points in the environment which may not accurately or

2018-08-30

IML
└Video games
 └Valve bots

Page 12 :

1 Classical approaches
- Mesh
- Graph

2 Video games
- Quake 3 Arena bots
- Counter-Strike bots
- Unreal Tournament's series bots
- Valve bots

3 GNG

4 SGNG

Page 13 :

Graph model

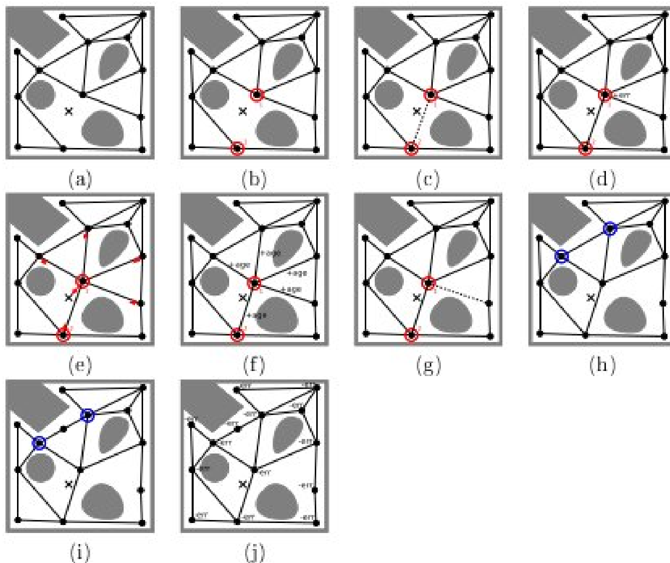
- ▷ incremental learning
- ▷ node
  - ◇ position (x, y, z)
  - ◇ cumulated error (measures how well the node represents its surroundings)
- ▷ edge
  - ◇ links two nodes
  - ◇ an age informing us when it was last activated.
- ▷ Principle : modify its graph for each input of the teacher's position in order to alter the graph to match the teacher's position. The model can add or remove nodes and edges if they do not fit the behavior and change the position of the nodes to better represent the teacher's position.

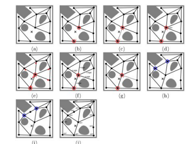
Page 14 :

GNG - algorithm

while *Number of nodes* $\leq N_{max}$ **do**
  Get input position (a);
  Pick the closest (n_1) and the second closest nodes (n_2) (b);
  Create edge between n_1 and n_2 (c).;
  If an edge already existed, reset its age to 0.;
  Increase the error of n_1 (d);
  Move n_1 and its neighbors toward the input (e);
  Increase the age of all the edges emanating from n_1 by 1 (f);
  Delete edges exceeding a certain age (g);
  if *Iteration number is a multiple of* η **then**
    Find the maximum error node n_{max};
    Find the maximum error node n_{max2} among the neighbor of n_{max} (h);
    Insert node between n_{max} and n_{max2} (i);
    Decrease the error of n_{max} and n_{max2};
  end
  Decrease each node's error by a small amount (j);
end

Page 15 :





Page 16 :

```
import numpy as np
from scipy import spatial
import networkx as nx
import matplotlib.pyplot as plt
from sklearn import decomposition

class GrowingNeuralGas:

    def __init__(self, input_data):
        self.network = None
        self.data = input_data
        self.units_created = 0

    def find_nearest_units(self, observation):
        distance = []
        for u, attributes in self.network.nodes(data=True):
            vector = attributes['vector']
            dist = spatial.distance.euclidean(vector, observation)
            distance.append((u, dist))
        distance.sort(key=lambda x: x[1])
        ranking = [u for u, dist in distance]
        return ranking

    def prune_connections(self, a_max):
        for u, v, attributes in self.network.edges(data=True):
            if attributes['age'] > a_max:
                self.network.remove_edge(u, v)
        for u in self.network.nodes():
            if self.network.degree(u) == 0:
                self.network.remove_node(u)
```


Page 17 :


```
    def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=
      False):
        # logging variables
        accumulated_local_error = []
        global_error = []
        network_order = []
        network_size = []
        total_units = []
        self.units_created = 0
        # 0. start with two units a and b at random position w_a and w_b
        w_a = [np.random.uniform(-2, 2) for _ in range(np.shape(self.data)[1])]
        w_b = [np.random.uniform(-2, 2) for _ in range(np.shape(self.data)[1])]
        self.network = nx.Graph()
        self.network.add_node(self.units_created, vector=w_a, error=0)
        self.units_created += 1
        self.network.add_node(self.units_created, vector=w_b, error=0)
        self.units_created += 1
```


Page 18 :

```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=
     False):
    ...
    # 1. iterate through the data
    sequence = 0
    for p in range(passes):
        print('   Pass #%d' % (p + 1))
        np.random.shuffle(self.data)
        steps = 0
        for observation in self.data:
            # 2. find the nearest unit s_1 and the second nearest unit s_2
            nearest_units = self.find_nearest_units(observation)
            s_1 = nearest_units[0]
            s_2 = nearest_units[1]
            # 3. increment the age of all edges emanating from s_1
            for u, v, attributes in self.network.edges_iter(data=True,
                 nbunch=[s_1]):
                self.network.add_edge(u, v, age=attributes['age']+1)
            # 4. add the squared distance between the observation and the
                 nearest unit in input space
            self.network.node[s_1]['error'] += spatial.distance.euclidean(
                 observation, self.network.node[s_1]['vector'])**2
```


Page 19 :


```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=
     False):
   ...
            # 5 .move s_1 and its direct topological neighbors towards the
                 observation by the fractions
            #    e_b and e_n, respectively, of the total distance
            update_w_s_1 = e_b * (np.subtract(observation, self.network.node
                 [s_1]['vector']))
            self.network.node[s_1]['vector'] = np.add(self.network.node[s_1
                 ]['vector'], update_w_s_1)
            update_w_s_n = e_n * (np.subtract(observation, self.network.node
                 [s_1]['vector']))
            for neighbor in self.network.neighbors(s_1):
                self.network.node[neighbor]['vector'] = np.add(self.network.
                     node[neighbor]['vector'], update_w_s_n)

            # 6. if s_1 and s_2 are connected by an edge, set the age of
                 this edge to zero
            #    if such an edge does not exist create it
            self.network.add_edge(s_1, s_2, age=0)
            # 7. remove edges with an age larger than a_max
            #    if this results in units having no emanating edges, remove
                 them as well
            self.prune_connections(a_max)
```


Page 20 :

```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=
      False):
    ...

            # 8. if the number of steps so far is an integer multiple of
                  parameter l, insert a new unit
            steps += 1
            if steps % l == 0:
                if plot_evolution:
                    self.plot_network('visualization/sequence/' + str(
                         sequence) + '.png')
                sequence += 1
                # 8.a determine the unit q with the maximum accumulated
                     error
                q = 0
                error_max = 0
                for u in self.network.nodes_iter():
                    if self.network.node[u]['error'] > error_max:
                        error_max = self.network.node[u]['error']
                        q = u
                # 8.b insert a new unit r halfway between q and its neighbor
                      f with the largest error variable
                f = -1
                largest_error = -1
                for u in self.network.neighbors(q):
                    if self.network.node[u]['error'] > largest_error:
                        largest_error = self.network.node[u]['error']
                        f = u
                w_r = 0.5 * (np.add(self.network.node[q]['vector'], self.
                     network.node[f]['vector']))
                r = self.units_created
                self.units_created += 1
```


2018-08-30

IML
└GNG

Page 21 :


```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=
      False):
    ...
                # 8.c insert edges connecting the new unit r with q and f
                #      remove the original edge between q and f
                self.network.add_node(r, vector=w_r, error=0)
                self.network.add_edge(r, q, age=0)
                self.network.add_edge(r, f, age=0)
                self.network.remove_edge(q, f)
                # 8.d decrease the error variables of q and f by multiplying
                      them with a
                #      initialize the error variable of r with the new value
                     of the error variable of q
                self.network.node[q]['error'] *= a
                self.network.node[f]['error'] *= a
                self.network.node[r]['error'] = self.network.node[q]['error'
                     ]
            # 9. decrease all error variables by multiplying them with a
                  constant d
            error = 0
            for u in self.network.nodes_iter():
                error += self.network.node[u]['error']
            accumulated_local_error.append(error)
            network_order.append(self.network.order())
            network_size.append(self.network.size())
            total_units.append(self.units_created)
            for u in self.network.nodes_iter():
                self.network.node[u]['error'] *= d
                if self.network.degree(nbunch=[u]) == 0:
                    print(u)
        global_error.append(self.compute_global_error())
```


2018-08-30

IML
└GNG

Page 22 :

```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=
      False):
   ...
    plt.clf()
    plt.title('Accumulated␣local␣error')
    plt.xlabel('iterations')
    plt.plot(range(len(accumulated_local_error)), accumulated_local_error)
    plt.savefig('visualization/accumulated_local_error.png')
    plt.clf()
    plt.title('Global␣error')
    plt.xlabel('passes')
    plt.plot(range(len(global_error)), global_error)
    plt.savefig('visualization/global_error.png')
    plt.clf()
    plt.title('Neural␣network␣properties')
    plt.plot(range(len(network_order)), network_order, label='Network␣order'
        )
    plt.plot(range(len(network_size)), network_size, label='Network␣size')
    plt.legend()
    plt.savefig('visualization/network_properties.png')
```


Page 23 :


```
def plot_network(self, file_path):
    plt.clf()
    plt.scatter(self.data[:, 0], self.data[:, 1])
    node_pos = {}
    for u in self.network.nodes_iter():
        vector = self.network.node[u]['vector']
        node_pos[u] = (vector[0], vector[1])
    nx.draw(self.network, pos=node_pos)
    plt.draw()
    plt.savefig(file_path)

def number_of_clusters(self):
    return nx.number_connected_components(self.network)
```


Page 24 :

```
def cluster_data(self):
    unit_to_cluster = np.zeros(self.units_created)
    cluster = 0
    for c in nx.connected_components(self.network):
        for unit in c:
            unit_to_cluster[unit] = cluster
        cluster += 1
    clustered_data = []
    for observation in self.data:
        nearest_units = self.find_nearest_units(observation)
        s = nearest_units[0]
        clustered_data.append((observation, unit_to_cluster[s]))
    return clustered_data

def reduce_dimension(self, clustered_data):
    transformed_clustered_data = []
    svd = decomposition.PCA(n_components=2)
    transformed_observations = svd.fit_transform(self.data)
    for i in range(len(clustered_data)):
        transformed_clustered_data.append((transformed_observations[i],
             clustered_data[i][1]))
    return transformed_clustered_data
```


Page 25 :


```
def plot_clusters(self, clustered_data):
    number_of_clusters = nx.number_connected_components(self.network)
    plt.clf()
    plt.title('Cluster␣affectation')
    color = ['r', 'b', 'g', 'k', 'm', 'r', 'b', 'g', 'k', 'm']
    for i in range(number_of_clusters):
        observations = [observation for observation, s in clustered_data if
             s == i]
        if len(observations) > 0:
            observations = np.array(observations)
            plt.scatter(observations[:, 0], observations[:, 1], color=color[
                 i], label='cluster␣#'+str(i))
    plt.legend()
    plt.savefig('visualization/clusters.png')

def compute_global_error(self):
    global_error = 0
    for observation in self.data:
        nearest_units = self.find_nearest_units(observation)
        s_1 = nearest_units[0]
        global_error += spatial.distance.euclidean(observation, self.network
             .node[s_1]['vector'])**2
    return global_error
```

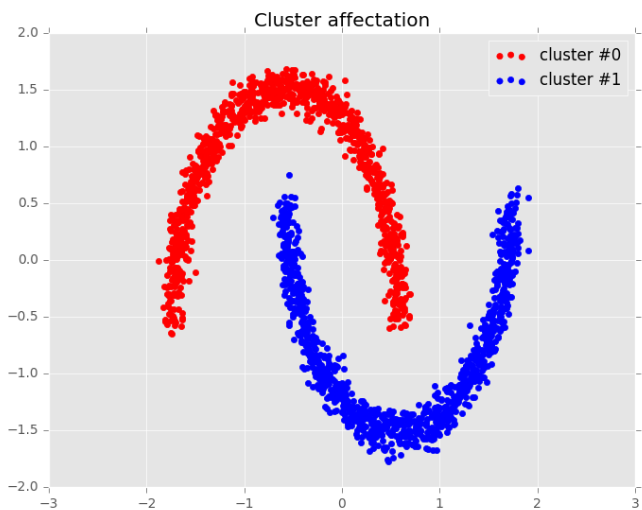

Page 26 :

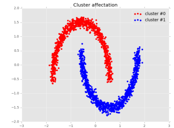
example

```
from gng import GrowingNeuralGas
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

print('Generating data...')
data = datasets.make_moons(n_samples=2000, noise=.05)
data = StandardScaler().fit_transform(data[0])
print('Done.')
print('Fitting neural network...')
gng = GrowingNeuralGas(data)
gng.fit_network(e_b=0.1, e_n=0.006, a_max=10, l=200, a=0.5, d=0.995, passes=8,
     plot_evolution=True)
print('Found %d clusters.' % gng.number_of_clusters())
gng.plot_clusters(gng.cluster_data())
```

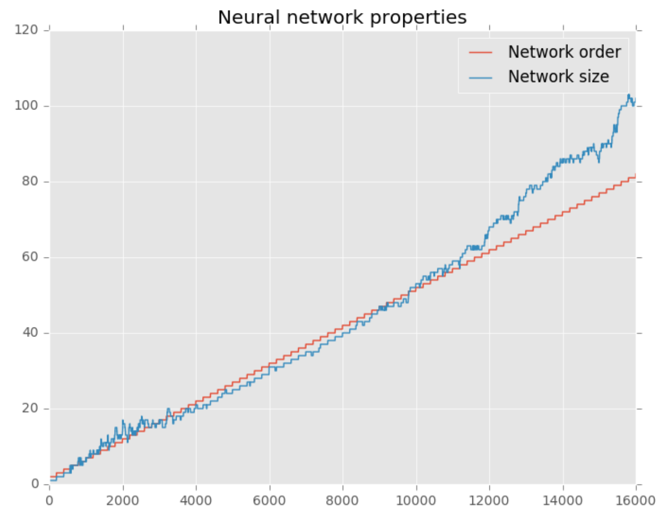

Page 27 :

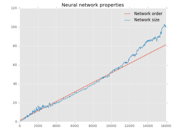
Preparing data...
Done.
Fitting neural network...
Pass #1
Pass #2
Pass #3
Pass #4
Pass #5
Pass #6
Pass #7
Pass #8
Found 2 clusters.



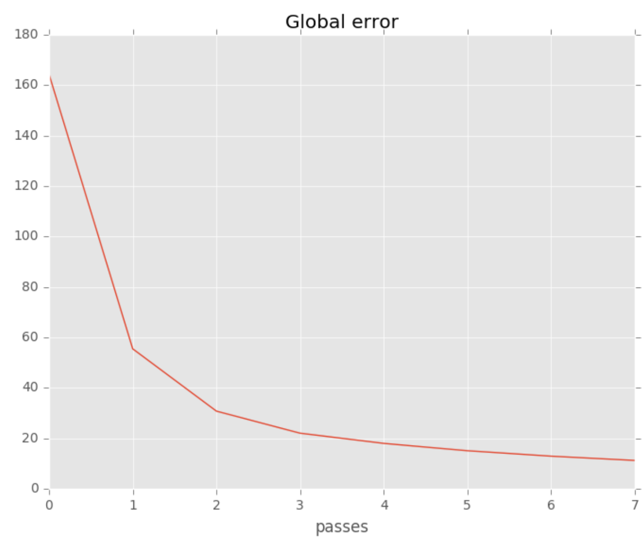


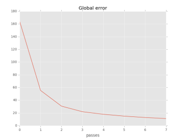
Page 28 :





Page 29 :





Page 30 :

Problems

- ▷ Fails to handle temporal series
- ▷ Grow up over the time (constant iteration insertion policy)

Page 31 :

1 Classical approaches
- Mesh
- Graph

2 Video games
- Quake 3 Arena bots
- Counter-Strike bots
- Unreal Tournament's series bots
- Valve bots

3 GNG

4 SGNG

Page 32 :

SGNG : Pro

- Grow up over the time : a node should be inserted when the error of a node is superior to a parameter $\overline{Err}$ (add nodes only in the area just visited and add nodes only when the network fails to fit the data suitably).

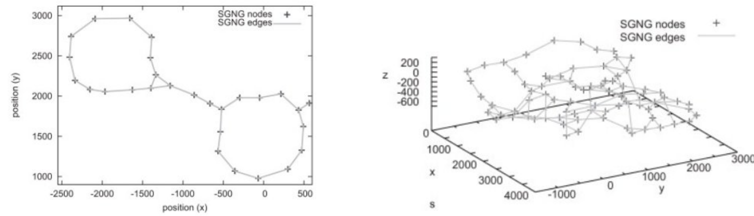
Page 33 :

SGNG - algorithm

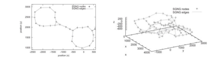
```
nodes ← {}
edges ← {}
while teacher plays do
    (x,y,z) ← teacher's position
    if |nodes| = 0 or 1 then
        nodes ← nodes ∪ {(x,y,z,error=0)}
    end if
    if |nodes| = 2 then
        edges ← {(nodes,age=0)}
    end if
    n1 ← closest((x,y,z),nodes)
    n2 ← secondClosest((x,y,z),nodes)
    edge ← edges ∪ {{n1,n2},age=0)}

    n1.error+=||(x,y,z)-n1||
    Attract n1 toward (x,y,z)
    ∀ edge ∈ edgesFrom(n1), edge.age++
    Delete edges older than Age (overlined)
    Attract neighbors(n1) toward (x,y,z)
    ∀ node ∈ nodes, node.error-=Err

    if n1.error > Err (overlined) then
        maxErrNei ← maxErrorNeighbour(n1)
        newNode ← between(n1,maxErrNei)
        n1.error/=2
        maxErrNei.error/=2
        newError ← n1.error+maxErrNei.error
        nodes ← nodes ∪ {(newNode,newError)}
    end if
```


Page 34 :

SGNG : Application to video games



https://www.youtube.com/watch?v=HSKkr4CQYr8



Page 35 :

SGNG : Application to video games

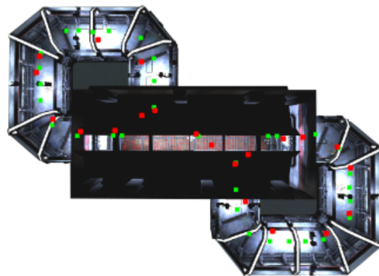
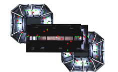


FIGURE – Comparison of nodes learned by the SGNG (in red) with the navigation points placed manually by the game developers (in green). The environment viewed from above is visible in the background.



Page 36 :

SGNG : Evaluation

Sensitivity

Sensitivity measures how successfully the SGNG represents the part of the environment the teacher used, which can be seen as true positives. $mindist(a, B)$ is the minimum distance between point a and the points in set B. We computed this measure using the following formula :

$$Sensitivity \propto \frac{1}{\sum_i mindist(NP_i, nodes)} \quad (1)$$

Where NP_i is the ith navigation point. The higher the value, the more sensitive the SGNG.

SGNG : Evaluation
Sensitivity
Sensitivity measures how successfully the SGNG represents the part of the environment the teacher used, which can be seen as true positives. $mindist(a, B)$ is the minimum distance between point a and the points in set B. We computed this measure using the following formula :
$Sensitivity \propto \frac{1}{\sum_i mindist(NP_i, nodes)}$ (1)
Where NP_i is the ith navigation point. The higher the value, the more sensitive the SGNG.

Page 37 :

SGNG : Evaluation

Specificity

Specificity measures how much the SGNG did not represent the part of the environment the teacher did not use, which can be seen as true negatives. We computed this measure using the following formula :

$$Specificity \propto \frac{Number\ of\ Nodes}{\sum_i mindist(Node_i, NPs)} \quad (2)$$

The higher the value, the more specific the SGNG.

SGNG : Evaluation
Specificity
Specificity measures how much the SGNG did not represent the part of the environment the teacher did not use, which can be seen as true negatives. We computed this measure using the following formula :
$Specificity \propto \frac{Number\ of\ Nodes}{\sum_i mindist(Node_i, NPs)}$ (2)
The higher the value, the more specific the SGNG.

Page 38 :

SGNG : Evaluation

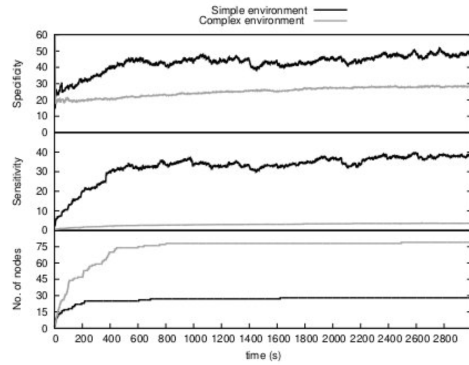
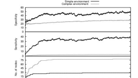


FIGURE – Time evolution of the SGNG number of nodes and the cumulated distance between the SGNG nodes and the navigation points.



Page 39 :

Classical approaches
Video games
GNG
SGNG

SGNG : Evaluation

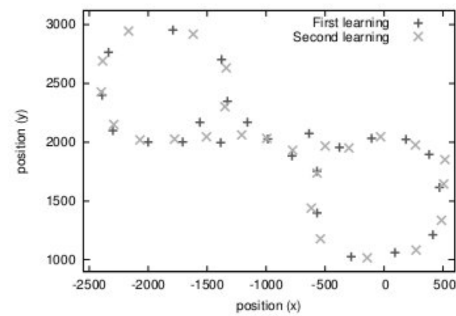
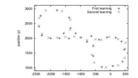


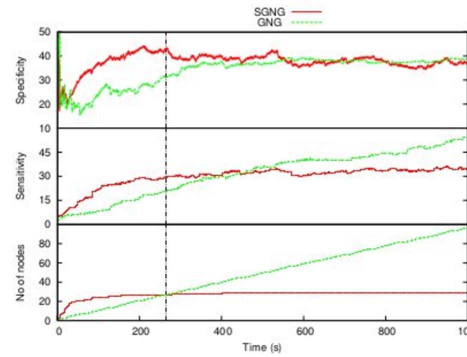
FIGURE – Comparison of two SGNG learned on the same environment after a very long training period of 10 hours.

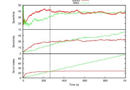
IML
└SGNG
└SGNG : Evaluation
2018-08-30



Page 40 :

SGNG : Evaluation





Page 41 :

Classical approaches
Video games
GNG
SGNG

SGNG : Référence

F. Tence, L. Gaubert, J. Soler, P. De Loor, and C. Buche.
Stable Growing Neural Gas : a Topology Learning Algorithm based on Player Tracking in Video Games.
Applied Soft Computing (ASC),
13(10) :4174-4184, 2013.

https://www.enib.fr/~buche/article/ASC_13.pdf

SGNG : Référence

F. Tence, L. Gaubert, J. Soler, P. De Loor, and C. Buche.
Stable Growing Neural Gas : a Topology Learning Algorithm based on Player Tracking in Video Games.
Applied Soft Computing (ASC).
13(10) :4174-4184, 2013.

https://www.enib.fr/~buche/article/ASC_13.pdf

Page 42 :

Navigation

IML

Cédric Buche

ENIB

30 août 2018

Navigation
IML
Cédric Buche
ENIB
30 août 2018

Page 43 :