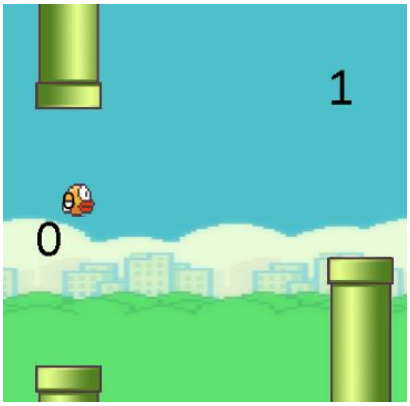
Lab practical: Neural Network



In this practical, we shall:

- Teach birds how to fly

Preparation:

- Zip files: SceneFlappy.h/.cpp, images

Explanation:

- Expand from 1 bird to a any number of birds
- Use Neural Network to help the birds make decision to jump
- Evolve the weights in the Neural Network

Exercise:

SceneTurn

1. SceneBase - add images

```
meshList[GEO_BG] = MeshBuilder::GenerateQuad("bg", Color(1, 1, 1));
meshList[GEO_BG]->textureID = LoadTGA("Image//background-day.tga");
meshList[GEO_SIDEBAR] = MeshBuilder::GenerateQuad("blackquad",
Color(0, 0, 0));
meshList[GEO_CHARACTER] = MeshBuilder::GenerateQuad("bird",
Color(1, 1, 1));
meshList[GEO_CHARACTER]->textureID =
LoadTGA("Image//yellowbird-downflap.tga");
meshList[GEO_PIPE] = MeshBuilder::GenerateQuad("pipe", Color(1, 1,
1));
meshList[GEO_PIPE]->textureID = LoadTGA("Image//pipe-green.tga");
```

2. Create a NeuralNode struct, with these attributes

```
std::vector<float> weights;
float output;
float z;
float alpha;
NNode(){}
~NNode(){}
```

3. In the same file or SceneFlappy, create this overloaded method to multiply weights and inputs

```
float operator*(const std::vector<float>& lhs, const
std::vector<float>& rhs);
```

4. GameObject.h - add necessary code for gameplay
    a. We use alive instead of active

b. Assume all GameObjects are active and GO_BIRD

```
enum GAMEOBJECT_TYPE
{
  GO_BIRD,
  GO_PIPE,
};
float score;
bool alive;
std::vector<NNode> hiddenNode;
NNode outputNode; //or you can combine into a single array
```

5. SceneFlappy.h - add any methods or variables that you need

6. SceneFlappy::Render() - render any additional information that you need

7. SceneFlappy::Init()
    a. Setup the nodes
    b. Initialize weights randomly - roll dice (-1.f, 1.f)

8. SceneFlappy::Restart()
    a. Reset the birds
    b. Mutate the birds' weights
        i. Mutate 50% bottom birds - reroll dice
        ii. Copy from strongest bird to weakest bird, mutate a little
        iii. Mutate rest a little (-0.01f, 0.01f)

9. SceneFlappy.cpp
    a. Implement Sigmoid()
    b. Implement Derivative()
    c. Implement FeedNN()

10. SceneFlappy::Update()
    a. Generating inputs for the NN

Bonus:

11. Build 3 layer network

12. Implement Learning by example using back-propagation

Assessment:

Component	Marks	Criteria
SceneFlappy		

2 layer NN	5	2 layer network with random weights that can allow birds to jump
Mutate	5	Learning by trial & error
Bonus		
3 layer NN	5	3 layers network
Learn	5	Learning by example, using back-propagation