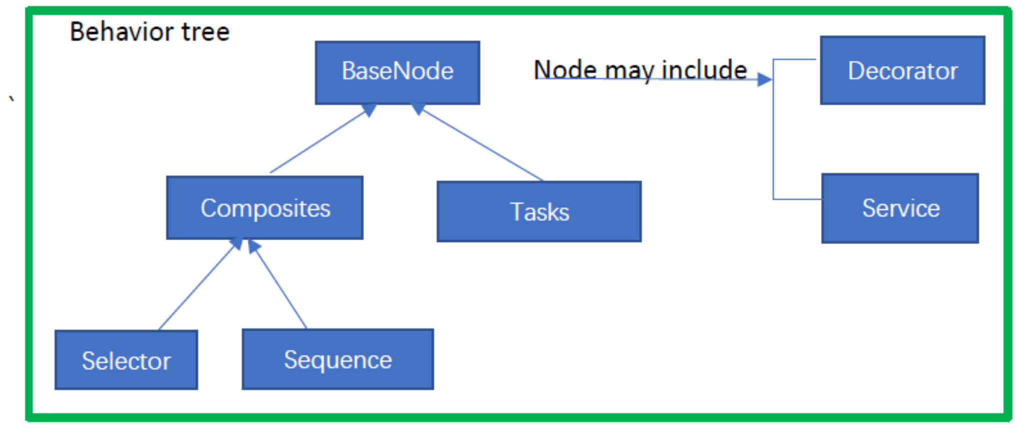
Engine Feature: Behavior Tree System

Download:

There are two parts in my system, one is behavior tree itself, one is blackboard which is used for behavior tree to store values.

Behavior tree: Make decision and execute tasks.



Behavior tree composites of many nodes. So I made a base node class for all the nodes, then the composite node and task node are two abstract class derived from base node. Node may include decorators and services, they are abstract base class in my system.

- Composite: Defines the root of a branch and the rules for how the branch is executed.
- Task: Leaf node that executes “tasks”.
- Decorator: Attached to nodes and make decisions on whether the node can be executed.
- Service: Attached to nodes and execute at a defined frequency as long as the branch is executed.

Selector composite, sequence composite and simple parallel are the specific composite class whose logic is frequently used in other behavior tree system.

- Selector: Executes the child nodes from first to last until one of the children succeeds. If one of them succeeds, the selector succeeds.
- Sequence: Executes the child nodes from first to last until one of the children fails. If one of them fails, the sequence fails.
- Simple parallel: Executes a task node and a composite node at the same time.

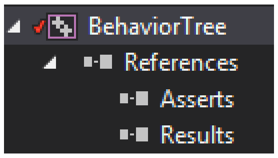
Blackboard: Store key values for behavior tree to use.

Blackboard composite of **cBlackboardSelector<T>** and a block of data memory. The type bool, int, float, Math::sVector, uintptr_t are supported. User can pass a **cBlackboardSelector<T>** to the blackboard to get the corresponding value, or pass a **cBlackboardSelector<T>** and a value to set the corresponding value.

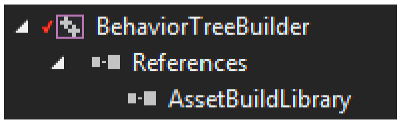
How to Use My Project

Setup Project

1. Download the project, add the two static libraries: BehaviorTree and BehaviorTreeBuilder into your project.
2. Add the reference as below:

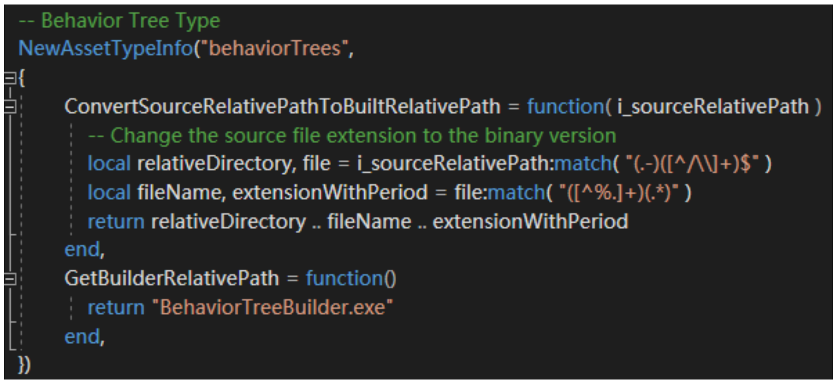


```
BehaviorTree
    References
        Asserts
        Results
```



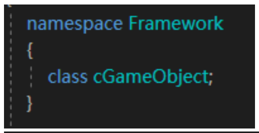
```
BehaviorTreeBuilder
    References
        AssetBuildLibrary
```

3. Add a new asset type in your AssetBuildFunctions.lua as below:

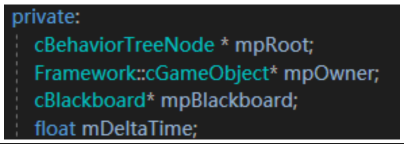


```
-- Behavior Tree Type
NewAssetTypeInfo("behaviorTrees",
{
    ConvertSourceRelativePathToBuiltRelativePath = function( i_sourceRelativePath )
        -- Change the source file extension to the binary version
        local relativeDirectory, file = i_sourceRelativePath:match( "(.-)([^/\]+)$" )
        local fileName, extensionWithPeriod = file:match( "([^%.]+)(.*)" )
        return relativeDirectory .. fileName .. extensionWithPeriod
    end,
    GetBuilderRelativePath = function()
        return "BehaviorTreeBuilder.exe"
    end,
})
```

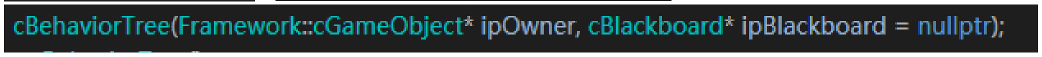
4. Open cBehaviorTree.h, you might need to change my Framework::cGameObject to what ever presents a game object in your project.



```
namespace Framework
{
    class cGameObject;
}
```



```
private:
    cBehaviorTreeNode * mpRoot;
    Framework::cGameObject* mpOwner;
    cBlackboard* mpBlackboard;
    float mDeltaTime;
```



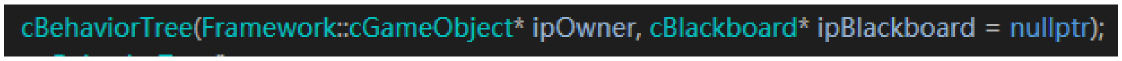
```
cBehaviorTree(Framework::cGameObject* ipOwner, cBlackboard* ipBlackboard = nullptr);
```

Use Behavior Tree

- Task
  Derived from **BehaviorTree_Task** then override the function **void ExcuteTask()**
- Composite
  Derived from **BehaviorTreeNode_Composite** then override the function **bool Execute()**
- Decorator
  Derived from **BehaviorTree_Decorator** then override the function **bool Check()**
- Service
  Derived from **BehaviorTree_Service** then override the function **void Run()**

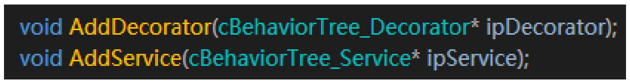
Setup behavior tree

1. Construct behavior tree object by cGameObject and cBlackboard



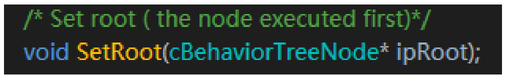
```
cBehaviorTree(Framework::cGameObject* ipOwner, cBlackboard* ipBlackboard = nullptr);
```

2. Create nodes and add decorators and services



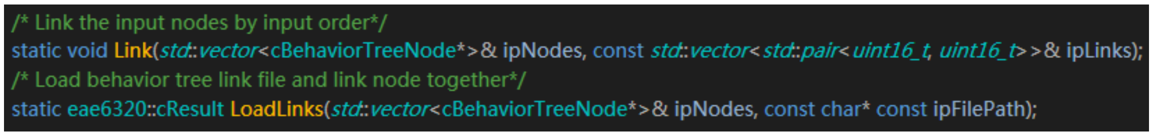
```
void AddDecorator(cBehaviorTree_Decorator* ipDecorator);
void AddService(cBehaviorTree_Service* ipService);
```

3. Set Root



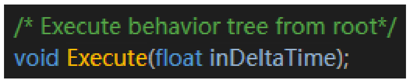
```
/* Set root ( the node executed first)*/
void SetRoot(cBehaviorTreeNode* ipRoot);
```

4. Link Nodes
   a. Push all nodes into a vector.
   b. Link it by a vector of pair of uint16_t or load links from file.



```
/* Link the input nodes by input order*/
static void Link(std::vector<cBehaviorTreeNode*>& ipNodes, const std::vector<std::pair<uint16_t, uint16_t>>& ipLinks);
/* Load behavior tree link file and link node together*/
static eae6320::cResult LoadLinks(std::vector<cBehaviorTreeNode*>& ipNodes, const char* const ipFilePath);
```

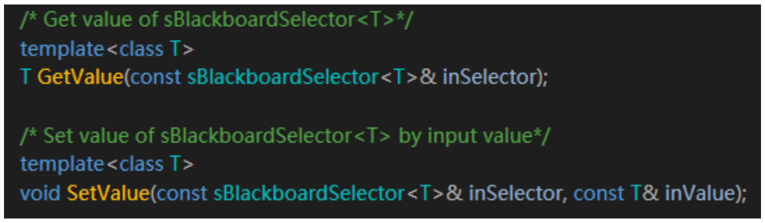
5. Execute every update



```
/* Execute behavior tree from root*/
void Execute(float inDeltaTime);
```

Behavior tree must cleanup by the end of program.

Use blackboard

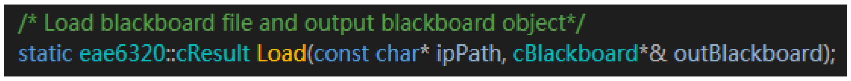


```
/* Get value of sBlackboardSelector<T>*/
template<class T>
T GetValue(const sBlackboardSelector<T>& inSelector);

/* Set value of sBlackboardSelector<T> by input value*/
template<class T>
void SetValue(const sBlackboardSelector<T>& inSelector, const T& inValue);
```

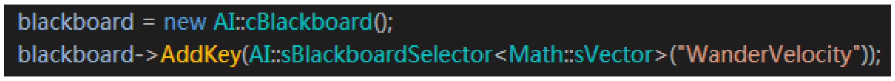
Setup Blackboard

1. Load from blackboard file.



```
/* Load blackboard file and output blackboard object*/
static eae6320::cResult Load(const char* ipPath, cBlackboard*& outBlackboard);
```

2. Add cBlackboardSelector<T> as keys

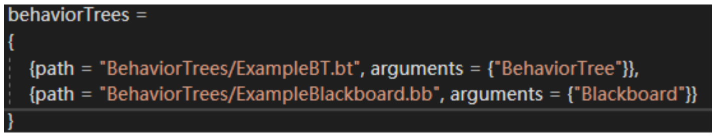


```
blackboard = new AI::cBlackboard();
blackboard->AddKey(AI::sBlackboardSelector<Math::sVector>("WanderVelocity"));
```

Human-readable files

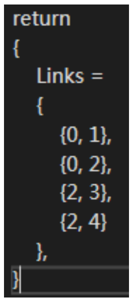
There are two types of human-readable files in my project. One is to link all the nodes in behavior tree, one is used to setup blackboard keys.

In AssetsToBuild.lua, add your human-readable file path as below, the argument is used to tell builder what kind of file we want to build.



```
behaviorTrees =
{
  {path = "BehaviorTrees/ExampleBT.bt", arguments = {"BehaviorTree"}},
  {path = "BehaviorTrees/ExampleBlackboard.bb", arguments = {"Blackboard"}}
}
```

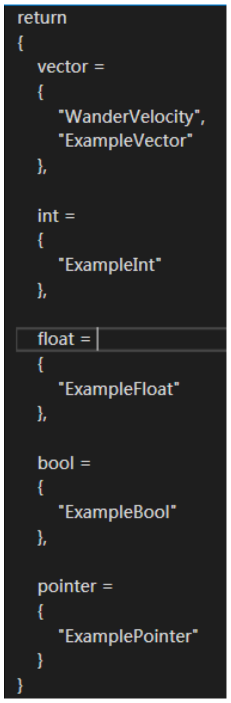
Behavior tree link file



```
return
{
  Links =
  {
    {0, 1},
    {0, 2},
    {2, 3},
    {2, 4}
  },
}
```

the number is the index of each node in the vector you store them.

Blackboard file



```
return
{
  vector =
  {
    "WanderVelocity",
    "ExampleVector"
  },

  int =
  {
    "ExampleInt"
  },

  float =
  {
    "ExampleFloat"
  },

  bool =
  {
    "ExampleBool"
  },

  pointer =
  {
    "ExamplePointer"
  }
}
```

vector, int, float, bool, pointer is the five type the blackboard supports. The string in each type table is the name of each variables.

Example:

There are several example tasks, decorators and services I've created, however it might not be able to run on your machine since I used some functions outside of the behavior tree project. I've moved all my examples and the relative files under the folder "Examples" for reference.

In `void eae6320::cMyGame::UpdateSimulationBasedOnTime(const float i_elapsedSecondCount_sinceLastUpdate)`, call execute function:

```
        mpBehaviorTree->Execute(i_elapsedSecondCount_sinceLastUpdate);
```

In eae6320::cResult eae6320::cMyGame::Initialize(), initialize as below:

```
// Initialize blackboard
	AI::cBlackboard* blackboard = nullptr;
	if (INITIALIZE_FROM_FILE)
	{
		AI::cBlackboard::Load("data/BehaviorTrees/ExampleBlackboard.bb", blackboard);
	}
	else
	{
		blackboard = new AI::cBlackboard();
		blackboard->AddKey(AI::sBlackboardSelector<Math::sVector>("WanderVelocity"));
	}


	// Initialize behavior tree
	mpBehaviorTree = new AI::cBehaviorTree(gameObject3, blackboard);

	AI::cTask_MoveToGameObject* task = new AI::cTask_MoveToGameObject(gameObject2);
	AI::cDecorator_GameObjectInSight* decorator = new 
AI::cDecorator_GameObjectInSight(gameObject2, 10);
	task->AddDecorator(decorator);

	AI::cBehaviorTree_Selector* selector = new AI::cBehaviorTree_Selector();
	AI::cBehaviorTree_Sequence* sequence = new AI::cBehaviorTree_Sequence();

	// wander
	AI::sBlackboardSelector<Math::sVector> bbSelector("WanderVelocity");
	AI::cTask_Wander* wander = new AI::cTask_Wander(bbSelector);
	AI::cBehaviorTree_Service* service = new AI::cService_WanderVelocity(4.0f, bbSelector);
	wander->AddService(service);

	AI::cTask_ToggleVisibility* toggleVisibility = new AI::cTask_ToggleVisibility();

	mpBehaviorTree->SetRoot(selector);

	std::vector<AI::cBehaviorTreeNode*> nodes;
	nodes.push_back(selector);
	nodes.push_back(task);
	nodes.push_back(sequence);
	nodes.push_back(wander);
	nodes.push_back(toggleVisibility);


	if (INITIALIZE_FROM_FILE)
	{
		AI::cBehaviorTree::LoadLinks(nodes, "data/BehaviorTrees/ExampleBT.bt");
	}
	else
	{
		// link behavior tree nodes
		std::vector<std::pair<uint16_t, uint16_t>> links;
		links.push_back(std::make_pair(0, 1));
		links.push_back(std::make_pair(0, 2));
		links.push_back(std::make_pair(2, 3));
		links.push_back(std::make_pair(2, 4));

		AI::cBehaviorTree::Link(nodes, links);
	}
```