Each Behavior Tree maintains a data structure called a Blackboard, that holds variables and values relevant to an AI agent. It can be regarded as the memory of the AI agent, and is used to decouple the data used in a Behavior Tree from its logic.

An agent's Blackboard can be accessed by any node to read/write data, during the execution of the Behavior Tree. However, since each agent maintains its own space of variables and values, Blackboard variables cannot be shared across different Behavior tree instances. The structure of an agent's Blackboard could be as follows:

Variable Name	Variable Value
Health	100.0f
Armor	55
Partner	“Entity-5”
Destination	(532, 23, 32)
ClosestCover	(543, 12, 32)
ClosestAttackPosition	(512, 21, 32)
…	…

Included within this page is a detailed look at the Blackboard interface, along with a C++ example showing how to use it.

Blackboard Interface

CRYENGINE's Blackboard uses a key-value data structure, where keys are of the type *BlackboardVariableId* which is just a wrapper for the variable name. A 32-bit CRC is used codify variable names and speed up key comparisons in the Blackboard.

Constructing a *BlackboardVariableId* is as simple as calling the constructor which takes a single argument.

BlackboardVariableId Constructor

```
//! Constructs a BlackboardVariableId from a variable name.
BlackboardVariableId(const char* _name);

// Could be used as follows.
const BlackboardVariableId myVariableId = BehaviorTree::
BlackboardVariableId("MyVariable");
```

The Blackboard interface exposes three functions:

Blackboard Interface

```
//! Creates a a new entry <id, value> in the Blackboard. If the given Id
already exists on the Blackboard, the value will be overwritten if, and
only if, the existing value type and provided type match.
//! If types do not match, value won't be modified and the function will
return false.
//! \param id Id of the variable.
//! \param value Value of the variable.
//! \return True if successfully set.
template<typename Type>
bool SetVariable(BlackboardVariableId id, const Type& value);

//! Gets the value of a variable by id if it exists on the Blackboard, and
the existing value type and provided type match.
//! \param id Id of the variable.
//! \param value Value of the variable.
//! \return True if Id exists and types match.
template<typename Type>
bool GetVariable(BlackboardVariableId id, Type& value) const;

//! Gets the Blackboard variable array.
//! \return BlackboardVariableArray containing all Blackboard entries <id,
value>.
//! \note Use this only if you need to iterate over Blackboard items. To
retrieve single items GetVariable(id, value) should be preferred instead.
const BlackboardVariableArray& GetBlackboardVariableArray() const;
```

To demonstrate how the above functions are used, the following code retrieves a variable *Health*, which is then set to **100.0f**.

SetVariable/GetVariable Example

```
Blackboard blackboard;

// Create a BlackboardVariableId from Health.
const BlackboardVariableId blackboardVariableHealth = BlackboardVariableId
("Health");

// Create an Health entry in the blackboard with a value of 100.0.
const bool variableSetSuccessfully = blackboard.SetVariable
(blackboardVariableHealth , 100.0f);

// Get the value from the blackboard into the health variable, which after
this, should have a value of 100.0.
float health;
const bool variableGetSuccessfuly = blackboard.GetVariable
(blackboardVariableHealth health);
```

Get The Blackboard

To retrieve the Blackboard of a specific Behavior Tree instance, the Blackboard is accessed through a *bla ckboard* class member in the *UpdateContext* parameter of any core Behavior Tree function *(OnInitialize, Update,* and *OnTerminate)*.

Get blackboard from UpdateContext

```
void OnInitialize(const UpdateContext& context)
{
        float health;
        Blackboard& blackboard = context.blackboard;

        // Access the blackboard.
        const bool variableGetSuccessfully = blackboard.GetVariable
("Health", health));

        // Execute specific node logic.
        // ...
}
```

To retrieve the blackboard of a specific AI agent, we use the function *GetBehaviorTreeBlackboard* from the *BehaviorTreeManager*, which accepts the Entity Id.

Get blackboard from BehaviorTreeManager

```
 gEnv->pAISystem->GetIBehaviorTreeManager()->GetBehaviorTreeBlackboard
(entityId);
```

Example

In the following example, a *MoveToPosition* node is created which allows us to move an AI agent to a specific position, that is stored in a variable in the Blackboard.

The node has a string data member *m_bbVariableNameDestination,* which specifies the name of the Blackboard variable that stores the destination of the agent. This variable name could be hard-coded, but ideally, it must be set on each tree to allow for flexibility.

Additionally, the value of this variable in the Blackboard could set by any other node. For example, we could have a node *CalculateCoverPosition* set the variable *CoverPosition*, which would then be used by the *MoveToPosition* node to move an agent to a cover position.

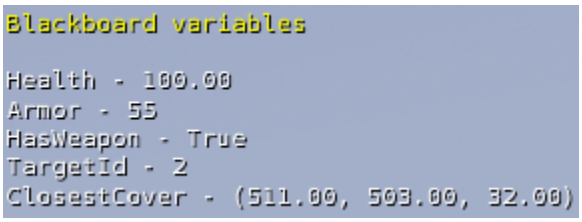
MoveToPosition Node

```
//! MoveToPosition node moves to the position stored in a variable in the
Behavior Tree Blackboard.
class MoveToPosition : public Action
{
        typedef Action BaseClass;
public:
        struct RuntimeData
        {
                MovementRequestID movementRequestID;
                Status pendingStatus;
                RuntimeData()
                        : movementRequestID(0)
                        , pendingStatus(Running)
                {}

                ~RuntimeData()
                {
                        if (this->movementRequestID)
                        {
                                gAIEnv.pMovementSystem-
>UnsuscribeFromRequestCallback(this->movementRequestID);
                                this->movementRequestID =
MovementRequestID();
                        }
                }

                void MovementRequestCallback(const MovementRequestResult&
```

```
result)
		{
			assert(this->movementRequestID == result.
requestID);
			this->movementRequestID = MovementRequestID();
			if (result == MovementRequestResult::
ReachedDestination)
			{
				this->pendingStatus = Success;
			}
			else
			{
				this->pendingStatus = Failure;
			}
		}
	};

public:
	virtual void OnInitialize(const UpdateContext& context) override
	{
		RuntimeData& runtimeData = GetRuntimeData<RuntimeData>
(context);
		Vec3 destination;
		if (context.blackboard.GetVariable(BlackboardVariableId
(m_bbVariableNameDestination), destination))
		{
			MovementRequest movementRequest;
			movementRequest.entityID = entityID;
			movementRequest.destination = destination;
			movementRequest.callback = functor(runtimeData,
&RuntimeData::MovementRequestCallback);
			movementRequest.style = m_movementStyle;
			runtimeData.movementRequestID = gAIEnv.
pMovementSystem->QueueRequest(movementRequest);

			if (runtimeData.movementRequestID.IsInvalid())
			{
				runtimeData.pendingStatus = Failure;
			}
		}
		else
		{
			runtimeData.pendingStatus = Failure;
		}
	}
	virtual Status Update(const UpdateContext& context) override
	{
		RuntimeData& runtimeData = GetRuntimeData<RuntimeData>
(context);
		return runtimeData.pendingStatus;
	}

	virtual void OnTerminate(const UpdateContext& context) override
	{
		// Actually, not necessary. Included to help illustrate
this example.
	}

private:
	string		m_bbVariableNameDestination;
	MovementStyle	m_movementStyle;
};
```

Debug

Blackboard entries can be displayed on screen during the execution of a Behavior Tree in Game Mode, by enabling the **ai_ModularBehaviorTreeDebugBlackboard** CVar. This however, only displays variables of types *int*, *bool*, *char*, *float*, *double, string, Vec2* and *Vec3* as shown.



Blackboard entries

Limitations

The current Blackboard implementation has three limitations:

- **Sandbox support:** Blackboard usage is not supported on Sandbox, meaning it can only be used via C++.
- **Shared variables:** Each Behavior Tree instance has its own Blackboard, meaning Blackboard variables cannot be shared between different instances.
- **Limited debug system**: Only Blackboard entries of a specific subset of types are shown when debugging.