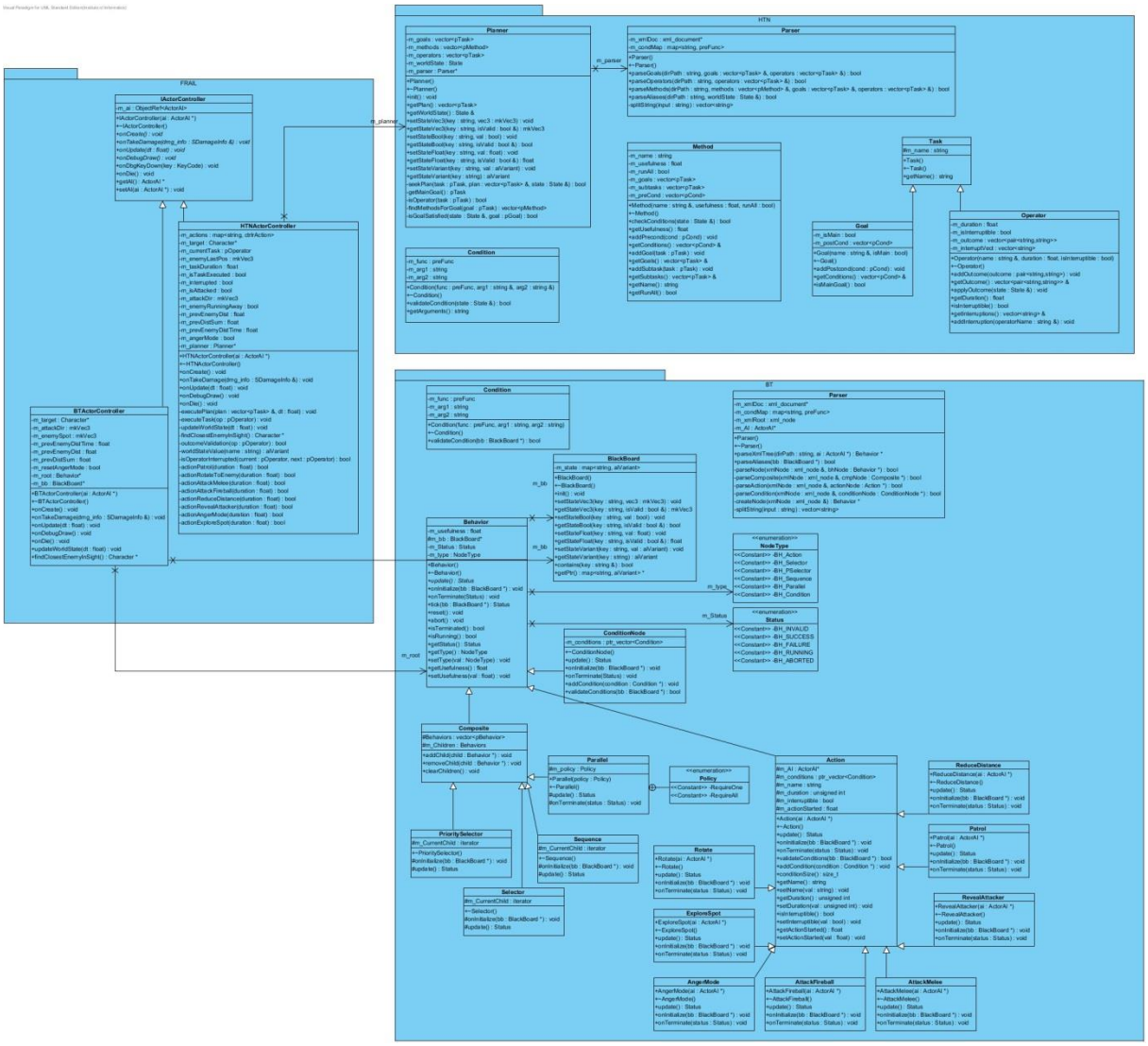
20.01.2014

User manual

AI architecture



Actor controllers are classes included into FRAIL framework which communicates with specific AI mechanism: Behavior Tree or Hierarchical Task Network.

Behavior Tree

To create BT-based AI you need to create controller file in FRAIL first.
All controllers' name must end with "**ActorController**" phrase (ex. BTActorController.cpp)!

1. In your fresh actor controller class, implement all abstract methods from inherited class **IActorController**.
2. Create members which will point to BT root node and blackboard.

3. In actor controller file in constructor body create new blackboard and initialize it by calling **init()** method.
4. In **onCreate()** method create local parser member and assign root node to result of parser's **parseXmlTree(std::string, ActorAI* ai)** method.
5. Call also **parseAliases(BT::BlackBoard *bb)** method to create user defined aliases.
6. In actor controller's method called **onUpdate(float dt)** you need to update blackboard values to let actions run properly.

   Example:
   ```
   m_bb->setStateFloat("ActorHealth", getAI()->getHealth());
   ```

7. Now you have to run **tick()** method in your root node object to start evaluating the tree. To do this, simply put **m_root->tick(m_bb)** in your **onUpdate(float dt)** method. You need to remember to pass blackboard object to the root node, so bt actions can transfer informations and evaluate preconditions.
8. The most important part is to create actions which can be called by their parent nodes. Go to **Actions.h** and create new class which extends **Action** class and implements its virtual methods.
9. The last step is to assign xml, action node's name to specific object. Navigate to **Parser.cpp** file and put new "**else if**" statement into **createNode(pugi::xmlNode& xmlNode)**.

   Example:
   ```
   else if(std::strcmp(xmlNode.attribute("name").value(),"MY_ACTION_NAME") == 0){
            result = new BT::MY_ACTION_CLASS(m_AI);
            result->setType(BH_Action);
            return result;
   }
   ```

Now you just need to put your actions into **tree.xml**, assign your controller name to AI "Preset" object in **ActorAI.json** file and run the game!

Hierarchical Task Network

1. Create new actor controller file with "**ActorController.cpp**" ending.
2. **Planner** object lets you get current plan based on previously designed methods, goals and operators. In your new actor controller you need to implement your own plan executor or copy existing one from **HTNActorConroller.cpp** file.
3. In your actor controller you need to implement abstract, inherited methods and create HTN::Planner object.
4. First initialize Planner by calling its **init()** method. (**init()** method call parser and initialize world state).
5. In your controller's **onUpdate(float dt)** method, you need to update world state and execute new plan by calling **m_planner->getPlan()**.
6. To create new action you need to assign your xml operator's name to unary function by placing it into m_actions std::map.

Example:

```
m_actions["opPatrol"] = &HTNActorController::actionPatrol;
```

7. Now declare that function in your actor controller's file and define it as:

```
bool HTNActorController::actionName(float duration){}
```

Sample HTN-based actor controller and BT-based actor controller implementation can be found in **HTNActorController.cpp** and **BTActorController.cpp** files.