Alterações feitas

- Primeiramente foi criada uma Queue para o signal
  - Essa queue vai segurar os holds que estão esperando por esse signal.
- Junto com essa queue foi criado então os métodos setter e getter para ela.

Alterações feitas

9 Componentes_implementados/Signal.h

```
@@ -1,6 +1,7 @@
 #ifndef SIGNAL_H
 #define SIGNAL_H
 #include "ModelComponent.h"
+#include "Queue.h"

 class Signal: public ModelComponent {
 public:
@@ -10,16 +11,22 @@ class Signal: public ModelComponent {
     virtual std::string show();
     static PluginInformation* GetPluginInformation();
     static ModelComponent* LoadInstance(Model* model, std::map<std::string, std::string>* fields);
+
     void setSignalName(std::string name) {
         signalName = name;
     }
     std::string getSignalName() {
         return signalName;
     }
+
+    std::string getQueueName() const;
+    void setQueueName(std::string _name) throw();
+
 private:
     int limit = 10;
     std::string signalName;
-
+    Queue* _holds_waiting_signal;
+
 protected:
     virtual void _execute(Entity* entity);
     virtual void _initBetweenReplications();
```

Alterações feitas

```
@@ -43,6 +42,20 @@ void Signal::_initBetweenReplications() {
43 42
44 43   }
45 44
   45 +
   46 + void Signal::setQueueName(std::string _name) throw() {
   47 +     Queue* queue = dynamic_cast<Queue*>(_model->getElementManager()->getElement(Util::TypeOf<Queue>(), _name));
   48 +     if (queue != nullptr) {
   49 +         _holds_waiting_signal = queue;
   50 +     } else {
   51 +         throw std::invalid_argument("Queue does not exist");
   52 +     }
   53 + }
   54 +
   55 + std::string Signal::getQueueName() const {
   56 +     return _holds_waiting_signal->getName();
   57 + }
   58 +
46 59   ModelComponent* Signal::LoadInstance(Model* model, std::map<std::string, std::string>* fields) {
47 60       Signal* newComponent = new Signal(model);
48 61       try {
```

Alterações feitas

- O método execute do signal foi alterado então para iterar sobre a queue que contém os holds aguardando.
- Ele realiza a mesma coisa de antes, simplesmente chama um método do hold que libera “limit” elementos da fila de espera do hold.

Alterações feitas

31 Componentes_implementados/Signal.cpp

```
@@ -20,16 +20,15 @@ inline bool instanceof(const T*) {
20 20      return std::is_base_of<Base, T>::value;
21 21  }
22 22
   23 + //TODO Colocar o instance of antes daquele cast ali.
   24 +
23 25  void Signal::_execute(Entity* entity) {
24    -     std::list<ModelComponent*>::iterator it = _model->getComponentManager()->begin();
25    -     for (; it != _model->getComponentManager()->end(); it++) {
26    -         auto component = *it;
27    -         if (instanceof<Hold>(*it)) {
28    -             Hold* h = ((Hold*)(*it));
29    -             if (h->getWaitForValueExpr() == signalName) {
30    -                 h->release_signal(limit);
31    -             }
32    -         }
   26 +     for(int i = 0; i < _holds_waiting_signal->size(); i++) {
   27 +         Waiting* waiting = _holds_waiting_signal->getAtRank(i);
   28 +
   29 +         auto component = waiting->getComponent();
   30 +         Hold* h = ((Hold*)(component));
   31 +         h->release_signal(limit);
33 32      }
```

Alterações feitas

- No hold, o que foi alterado foi simplesmente que quando o tipo deste hold for de esperar por um sinal, procura a fila com aquele nome, e se insere nela

Alterações feitas

3 Componentes_implementados/Hold.cpp

```
@@ -87,14 +87,17 @@ void Hold::_execute(Entity* entity) {
87  87            }
88  88        }
89  89        else if (_type == Type::WaitForSignal) {
    90  +         Queue* signal_queue = dynamic_cast<Queue*>(_model->getElementManager()->getElement(Util::TypeOf<Queue>(), _wait_fo
90  91            Waiting* waiting = new Waiting(entity, this, _model->getSimulation()->getSimulatedTime());
91  92            this->_queue->insertElement(waiting);
    93  +         signal_queue->insertElement(waiting);
92  94        }
93  95    }
94  96
95  97    void Hold::release_signal(int _limit) {
96  98          for(int i = 0; i < _queue->size(); i++) {
97  99                Waiting* waiting = _queue->getAtRank(i);
    100 +
98  101               _model->sendEntityToComponent(waiting->getEntity(), this->getNextComponents()->front(), 0.0);
99  102               if (i >= _limit) {
100 103                     break;
```