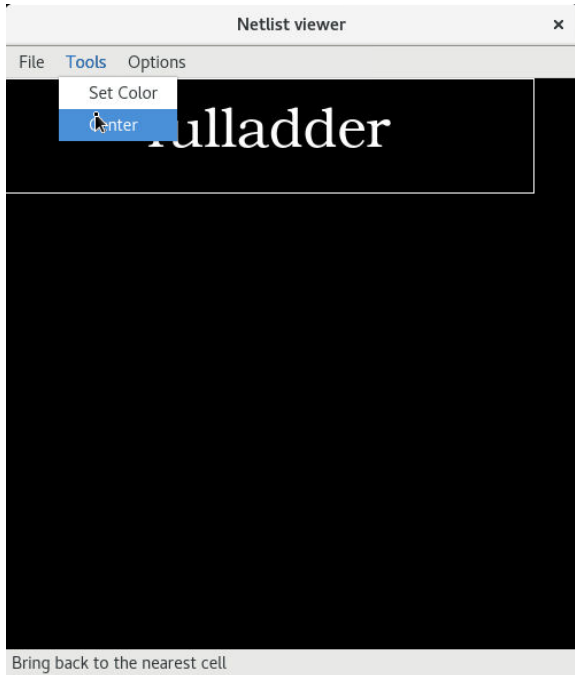
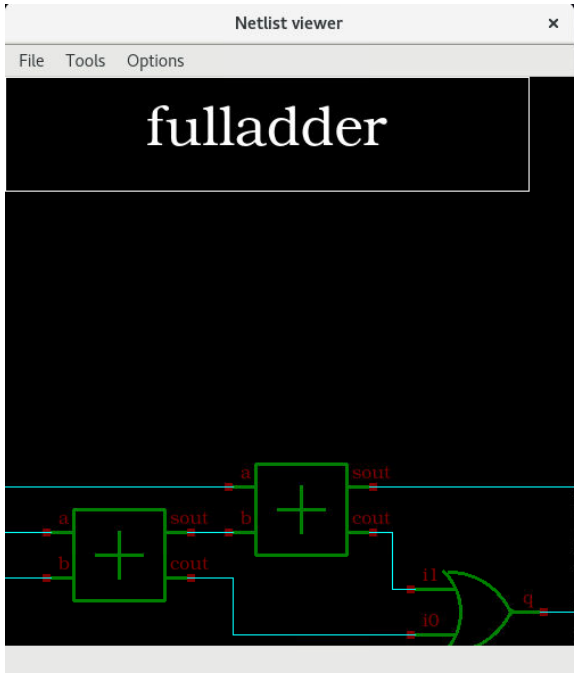


Ajouts supplémentaires au TME :

(La description des TME 8, 9 et 10 sont après la description des ajouts supplémentaire (à partir de la page 5))

Ce compte rendu a été rédigé en même temps que certains ajouts donc les images ne contiennent pas forcément tous les ajouts.

Le Center (dans CellViewer & CellWidget)	
<p>On est Perdu :</p>  <p>Bring back to the nearest cell</p>	<p>On retrouve la netlist après avoir appuyé :</p> 
<p>Permet de retrouver la netlist si on s'est perdu en nous ramenant vers elle dans le coin superieur gauche du widget. Si la netlist n'existe pas (vdd, gnd, etc...), ça nous ramène vers le centre.</p>	

Ajout supplémentaire aussi dans la fonction Open cell, si la cell n'existe pas encore, elle va la load puis l'ouvrir, sinon elle va juste l'ouvrir.

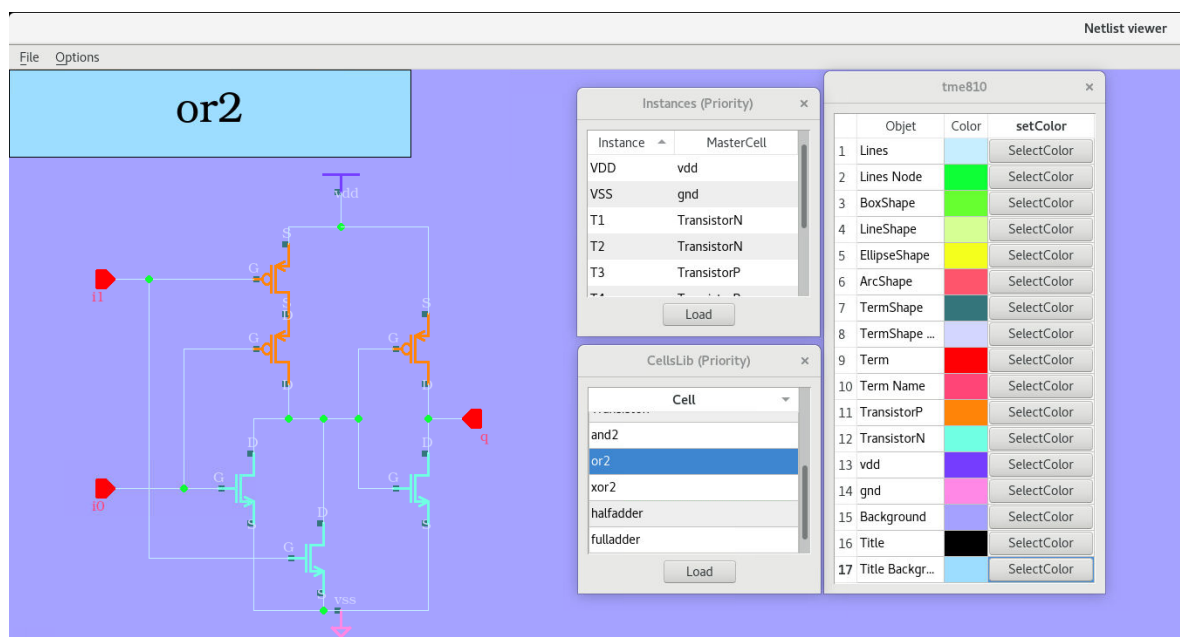
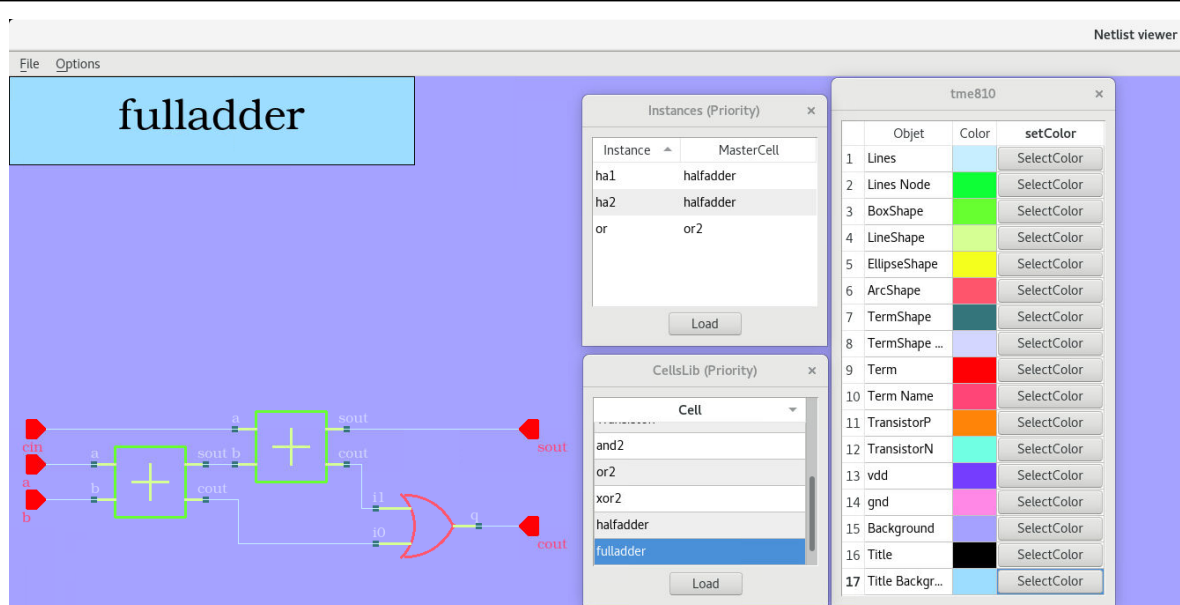
Donc le bouton Open cell, sert aussi à chargé les cellules pas encore chargé.

Si par exemple, on essaie de charger un fulladder et qu'on n'a pas encore chargé halfadder et or2, il faudra les chargées avant (Le terminal affichera une erreur qui préviendra l'utilisateur de vérifier si les modèles manquant ont été chargés).

Utile uniquement si le dossier n'est pas présent au moment où on lance l'application et qu'on rajoute le dossier après sans fermer l'application.

Dans le Main.cpp, il faudra enlever tous les load pour que ça fonctionne.

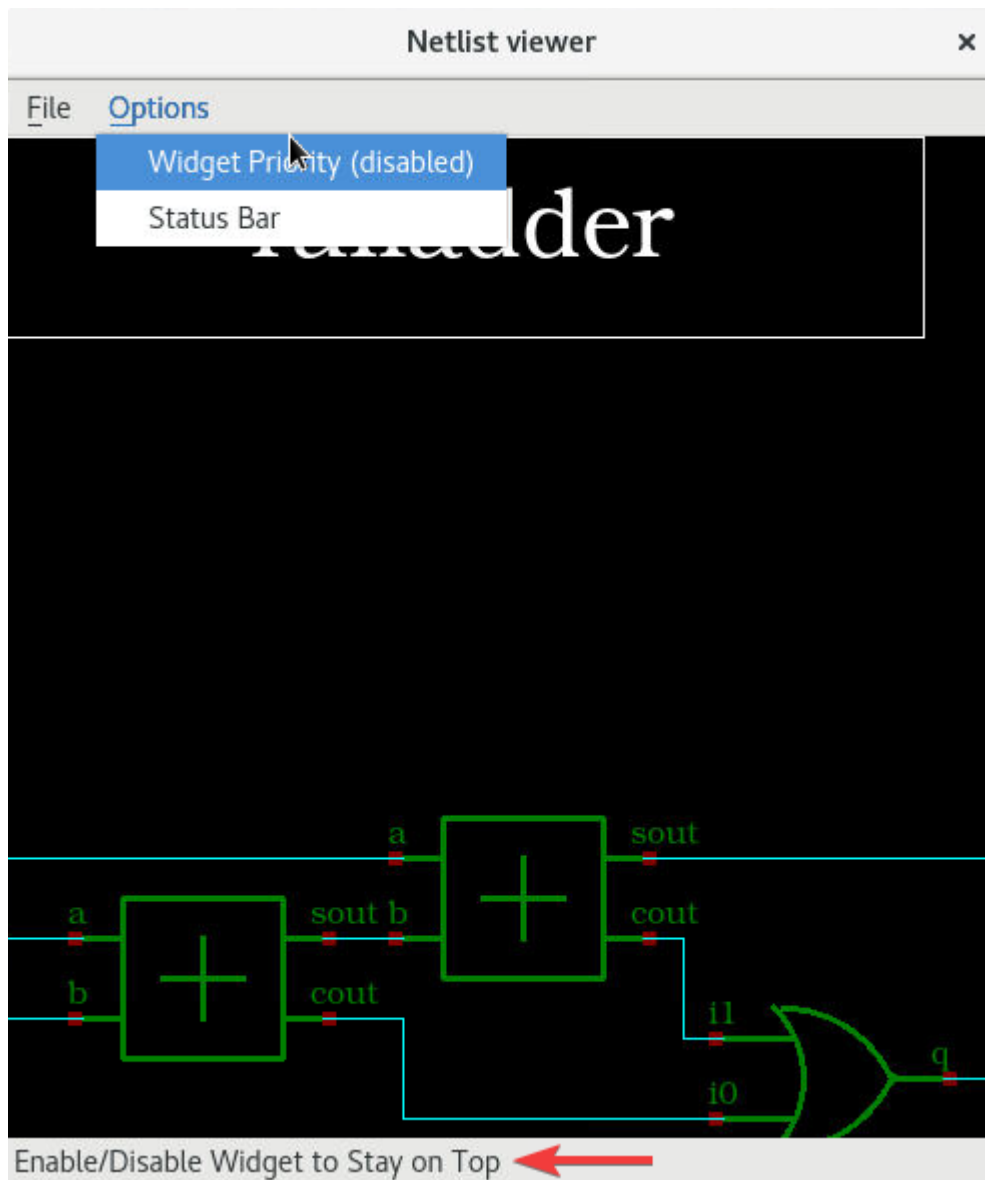
Le setColor (nouveau fichier color)



Permet de changer la couleur de pratiquement tout l'affichage de la netlist. Lorsqu'on change la couleur d'un transistor qui est constitué de LineShape, ça changera uniquement la couleur des transistors et pas des autres shapes qui ne sont pas des transistors. De même pour gnd et vdd.

Les deux images ci-dessus illustrent parfaitement que les couleurs des TermShape des transistors n'est pas lié à la couleur des TermShape des halfadder du fulladder.

La StatusBar (dans CellViewer)



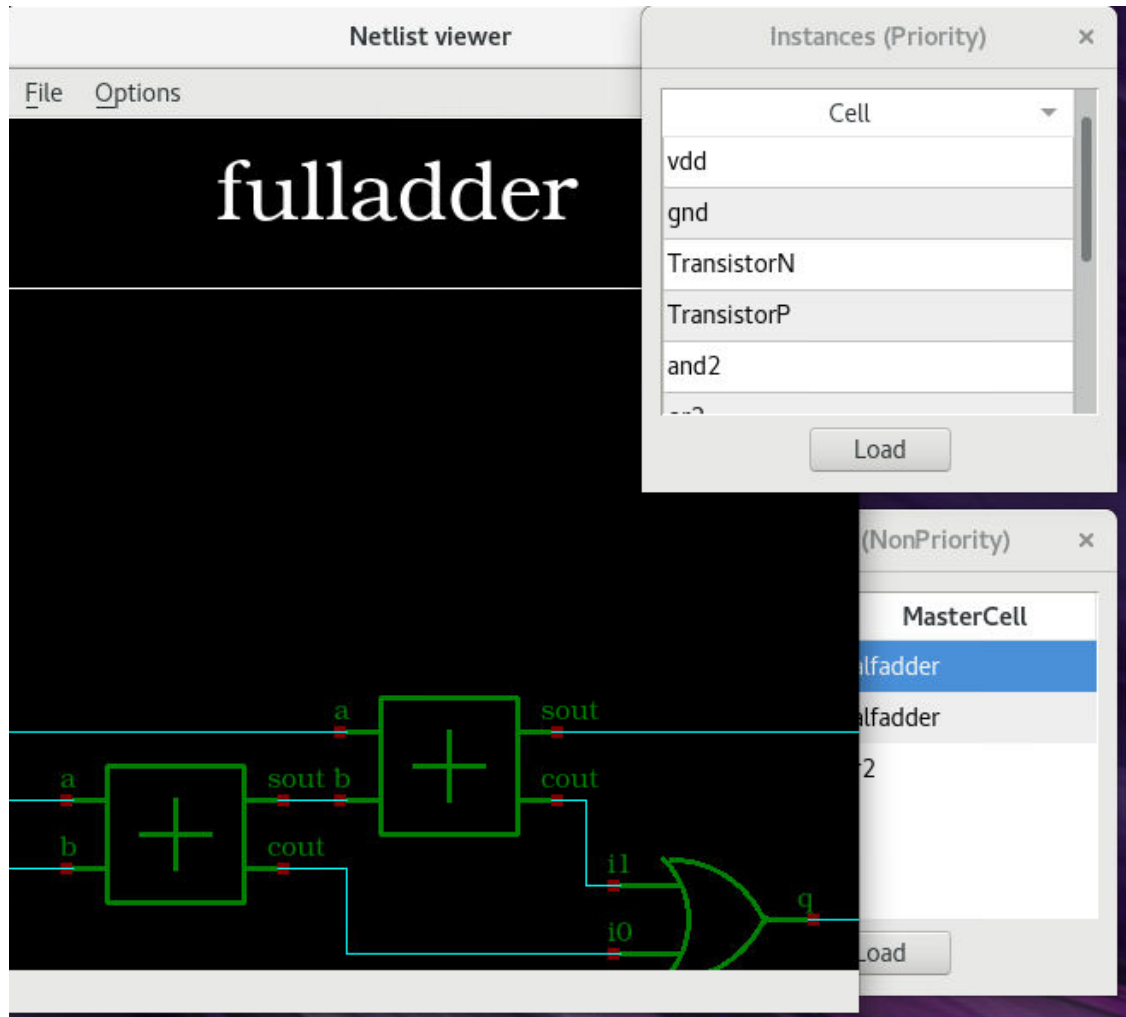
Permet d'afficher en bas du widget la StatusBar qui affiche le "setStatusTip" des différents choix de la barre de menu.

Car le setStatusTip n'affiche rien lorsqu'on survole les choix.

Le Widget Priority (dans CellViewer & InstancesWidget)

Permet d'afficher les widgets "InstancesWidget" et "CellsLib" toujours au premier plan pour les cas où on passe en plein écran, ça nous évite de perdre les widgets
L'option s'active ou se désactive dans les Options.

Par défaut, c'est non prioritaire. Fonctionne même quand les widgets sont ouverts



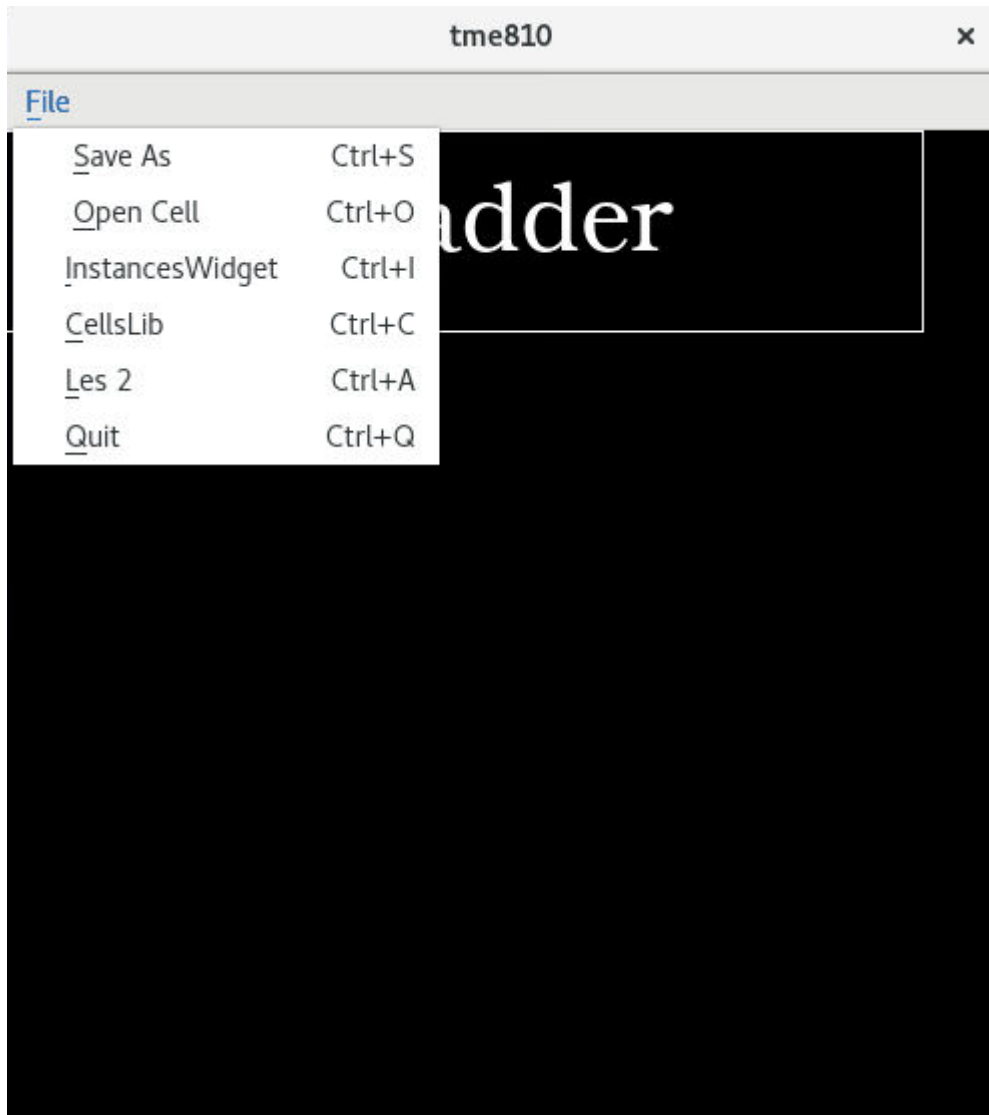
Ici, on voit comment la priorité marche, le widget Prioritaire est en avant plan et ne peut passer en arrière-plan tandis que le widget non prioritaire peut passer en arrière-plan.
(image non reproductible, car ils seront prioritaires ou non prioritaire ensemble)

Les widgets passeront même au-dessus de toutes les fenêtres (terminal, gestionnaire de fichier, etc...)

Question TME 8 : CellViewer, SaveCellDialog et OpenCellDialog

Le Widget CellViewer

Pour son constructeur, nous utilisons les méthodes de Qt pour créer le widget central ainsi que la barre de menu :

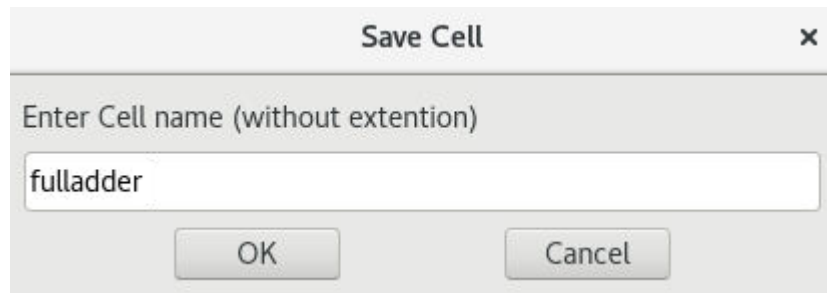


Normalement, lorsqu'on appuie sur chacune des options de la barre de menu, nous lancerons la méthode associée à l'option qui sera vue un peu plus bas.

(Le choix "Les 2" ouvre InstancesWidget et CellsLib en même temps)

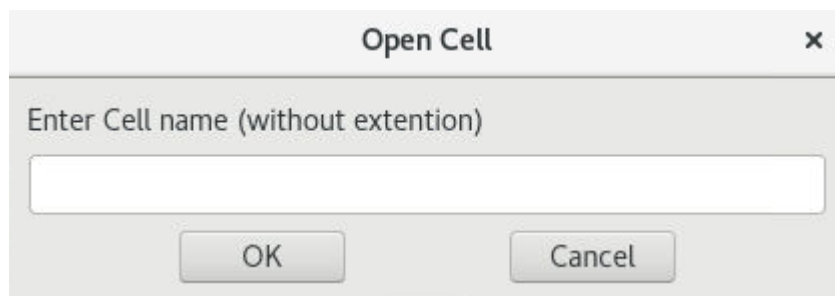
Le Widget SaveCellDialog

Pour son constructeur, nous utilisons les méthodes de Qt pour créer le widget suivant :



Le Widget OpenCellDialog

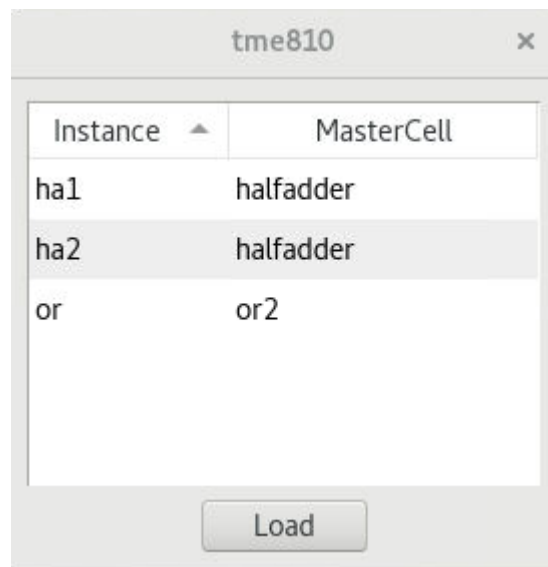
Avec le constructeur de ce widget, nous utilisons les méthodes de Qt pour créer le widget suivant :



Les méthodes d'OpenCellDialog sont les mêmes que celui de SaveCellDialog sauf pour la méthode run qui est une méthode statique :

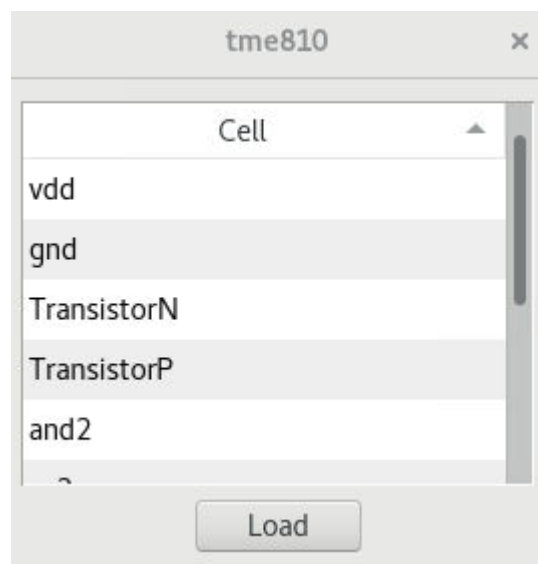
Question TME 9 :

InstancesWidget et InstancesModel



InstancesWidget se chargera de créer le widget et de l'affichage en récupérant les données nécessaires dans InstancesModel

CellsLib et CellModel



C'est la même chose que InstancesWidget, CellsLib gère la partie affichage du widget et CellsModel gère les données.

Question TME 10 : CellWidget

J'ai dû corriger une fonction dans "Instance.cpp" du TME7: void Instance::setPosition(int x, int y)

Avant :

```
void Instance::setPosition(int x, int y){
    position_.setX(x);
    position_.setY(y);
    for ( vector<Term*>::iterator iterm=terms_.begin() ; iterm !=
terms_.end() ; ++iterm ) {
        Point pts = owner_->getSymbol()->getTermPosition((*iterm));
        (*iterm)->setPosition( pts.getX() + x, pts.getY() + y);
    }
}

void Instance::setPosition(const Point& p){
    setPosition(p.getX(), p.getY()); //on lance la methode void
Instance::setPosition(int x, int y)
}
```

Je n'allais pas chercher les termshape, donc je n'allais pas chercher la position du connecteur associé au modèle, mais juste la position du model translaté, donc j'avais des fils qui partaient de partout.

Version corrigée :

```
void Instance::setPosition(int x, int y){
    position_.setX(x);
    position_.setY(y);
    for ( vector<Term*>::iterator iterm=terms_.begin() ; iterm !=
terms_.end() ; ++iterm ) {
        Shape* shape = getMasterCell()->getSymbol()->getTermShape(*iterm);
        TermShape* termshape = dynamic_cast<TermShape*>(shape);
        if(termshape){
            (*iterm)->setPosition(termshape->getX()+x,
termshape->getY()+y);
//(*iterm)->setPosition(getPosition().translate(termshape->getX(),
termshape->getY()));
        }
    }
}
```

Maintenant, on va bien chercher la bonne position, là où se trouve le connecteur, la partie commentée, c'est le code alternatif qui aurait pu être utilisé en utilisant les méthodes déjà présentes.

Pour l'affichage, j'ai décidé de mettre l'encadré qui contient le nom de la cellule qu'on observe tout le temps en haut à gauche.

L'affichage sera géré par la fonction `CellWidget::paintEvent` qui se chargera de lancer la fonction `CellWidget::query` qui fait l'affichage de nos cellules.

Les déplacements avec les flèches directionnelles du clavier sont fonctionnels pour pouvoir naviguer sur l'affichage.

Voici quelque capture d'écran de ce que nous obtenons :

