Soutenance LightBot - Projet POO

GENTY Laurent & DEVRIESERE Aymeric

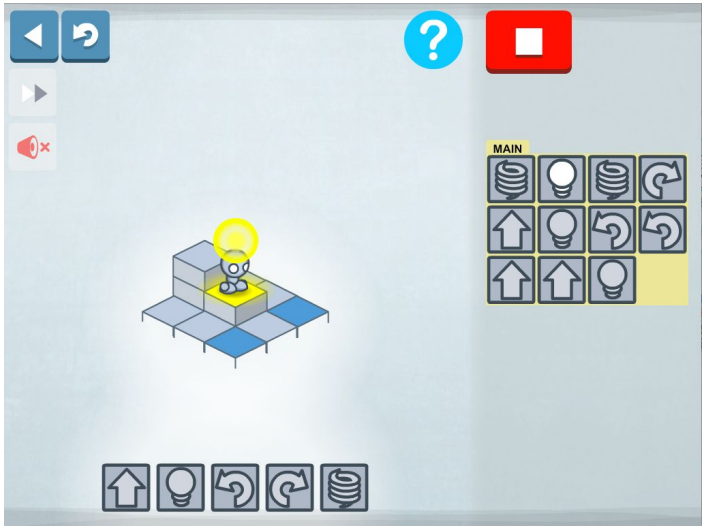
Sommaire

Principes du jeu : le LightBot

Jeu éducatif pour les enfants pour leur introduire la notion de programmation

Le robot doit allumer toutes les cases pour terminer un niveau

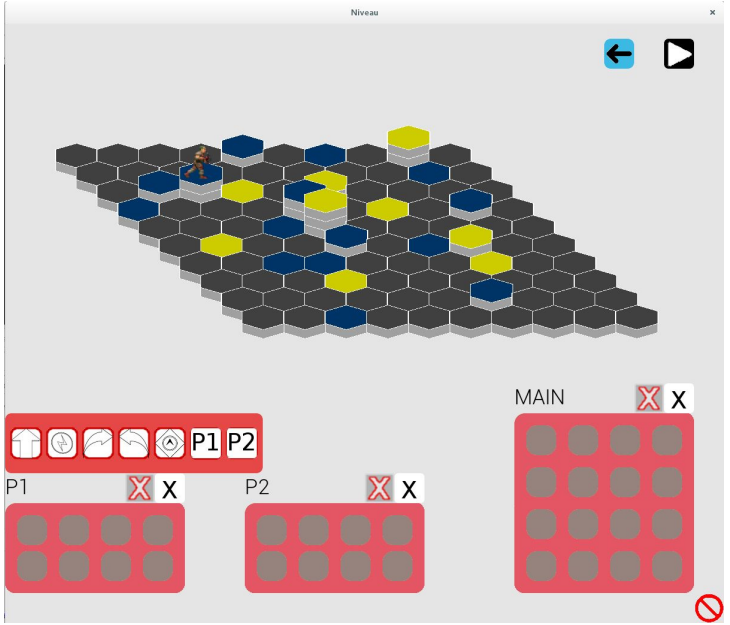
Construire une série d’instructions afin d’atteindre son objectif



Spécificités

Nous avons introduits différentes options :

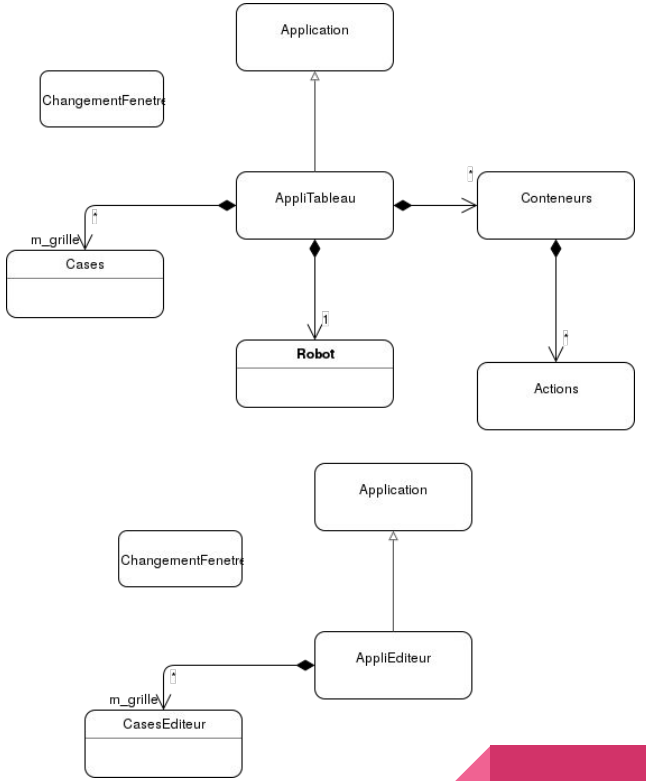
- système de hauteur (étages)
- vue isométrique (aspect de 3 dimensions)
- récursivité (boucles P1 dans P2 et P2 dans P1)
- édition de niveaux (sauvegarde et édition)



Conception générale

Construit sous forme de deux applications principales :

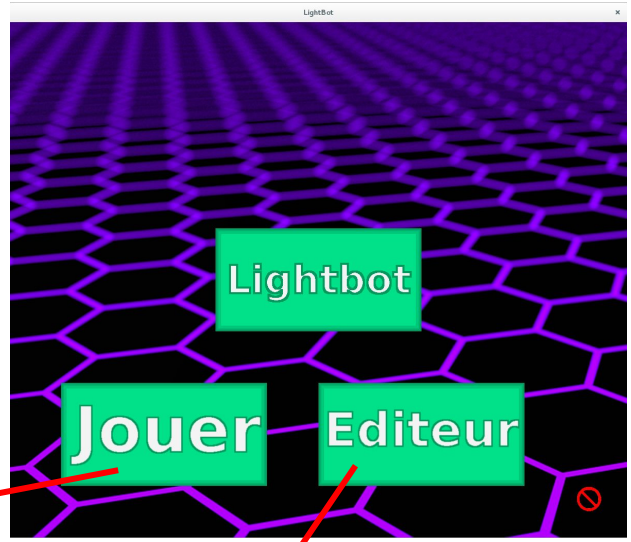
- AppliTableau
  - des conteneurs (qui contiennent des Actions)
  - un Robot
  - un tableau de Cases
- AppliEditeur
  - tableau de CasesEditeur



Différents écrans et ce qui fonctionne

Écran de menu :

- des boutons permettant de lancer des fenêtres différentes (sélection de niveaux ou édition)
- les écrans sont des classes Appli héritant de toutes d'Application
- classe ChangementFenetre permettant de basculer sur une autre

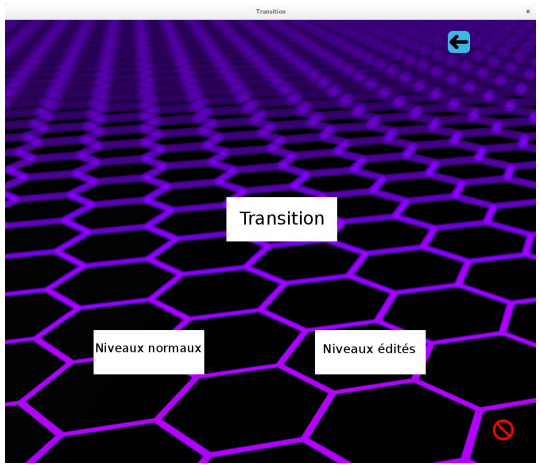


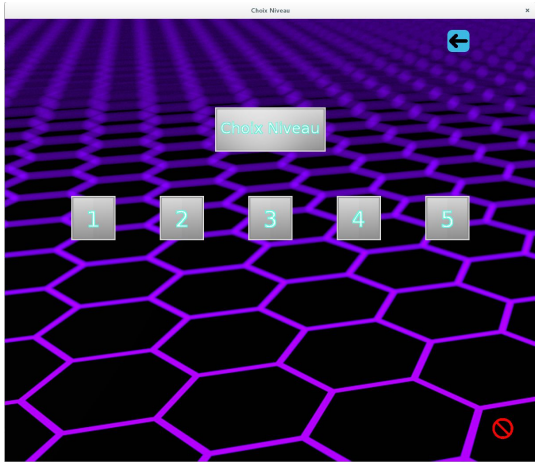
Choix des niveaux

Choix entre niveaux “normaux” et niveaux édités

Les niveaux :

- ❏ enregistrés dans des fichiers .txt
- ❏ on peut modifier les niveaux édités

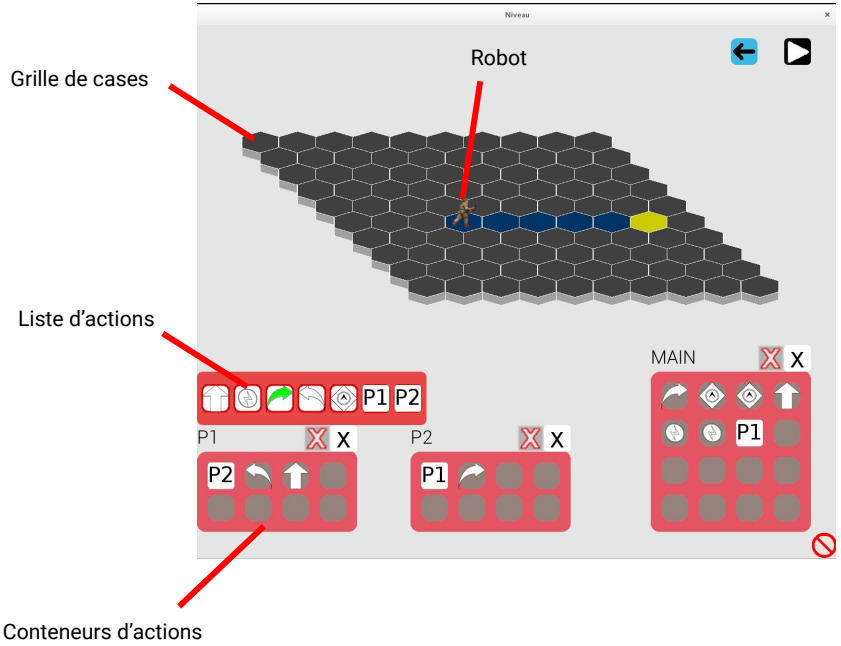




L’écran de jeu

Choisi de faire dans l’ordre :

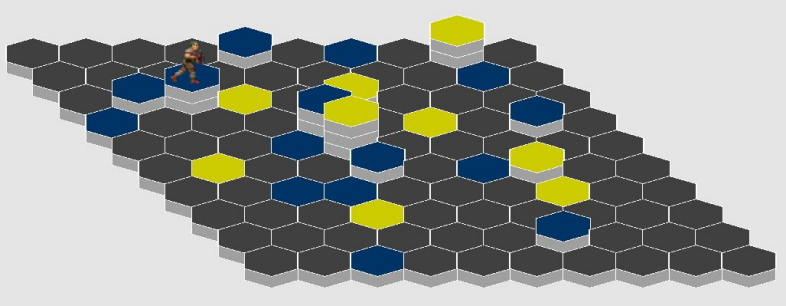
- ❑ la grille de cases
- ❑ les conteneurs d’actions
  - ❑ lancer les programmes
  - ❑ vérifier que la liste d’actions marche avec la récursivité
- ❑ Robot n’était pas une priorité !



Premier objectif : la grille de cases

La classe Cases :

- ❏ Réalisée de manière à être modifiable
- ❏ Réalisée à partir de Circle Shape
- ❏ Une case gère toute seule ses étages
- ❏ Les cases sont différentes selon leur type
- ❏ Aspect 2D isométrique



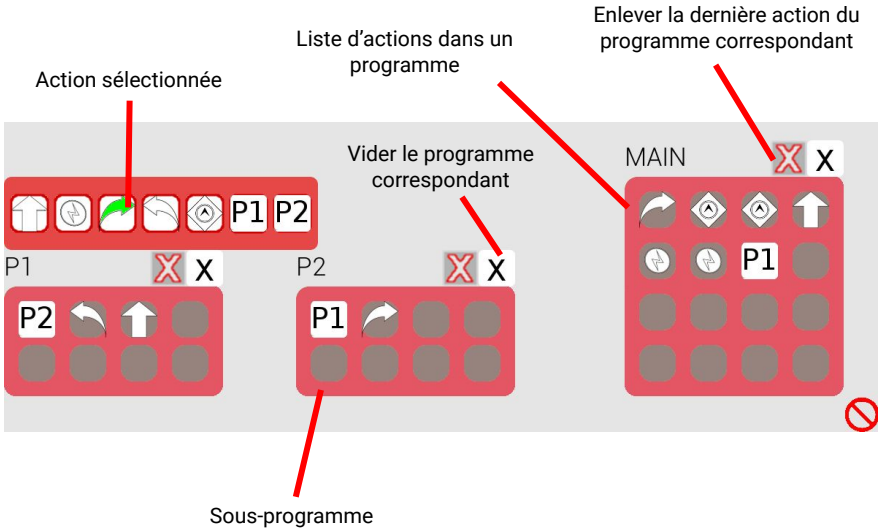
Les conteneurs et actions

Action :

- ❏ Etat correspondant à l'action précise
- ❏ booléen m_selec
- ❏ Ajouter des actions en cliquant dans un conteneur

Conteneurs :

- ❏ Vecteur d'actions
- ❏ Lancer les actions en séquence



Le “robot”

La classe Robot:

Les plus:

- ❏ Gère ses déplacement elle-même(switch)
- ❏ Gère son propre affichage
- ❏ Gère les étages
- ❏ Connaît sa direction
- ❏ Gère descente / montée

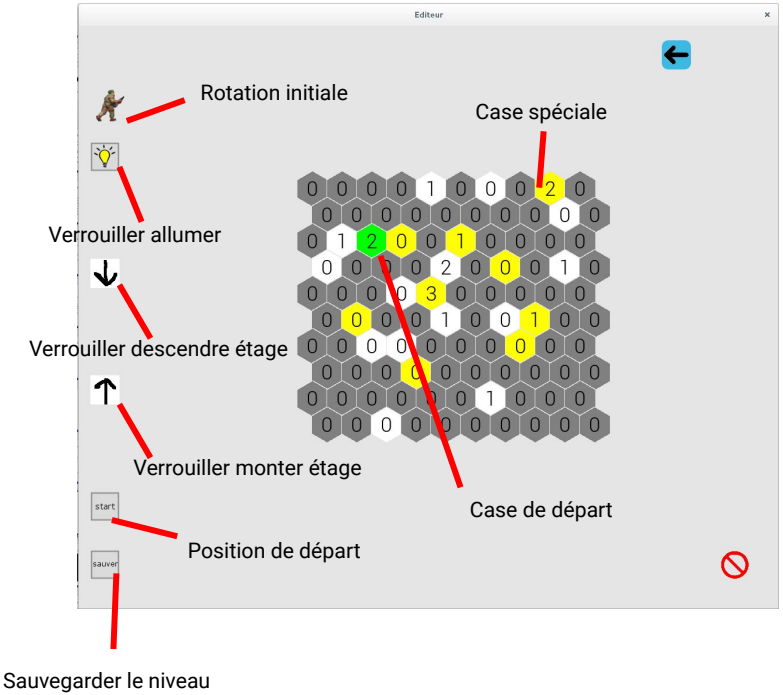
Les moins:

- ❏ Le sprite se multiplie

Editeur

La classe éditeur :

- ❏ Possède des états
- ❏ Positionne les cases en fonction du rayon
- ❏ Permet sauvegarde simple
- ❏ Permet de définir l'état des cases
- ❏ Animation pour sélectionner position initiale



Cases éditeur

- Affichent elle-mêmes leur texte
- Gèrent leur couleur (booléens)
- Peuvent être réinitialisé automatiquement
- Possèdent une fonction utile à la sauvegarde du niveau

Algorithme de déplacement : la simple récursivité !

```cpp
void Conteneur::executer(Robot *r, Conteneur *main, Conteneur *p1, Conteneur *p2, int programmeLance, bool &fini, sf::RenderWindow &window)
{
    //selon le programme qu'on lance soit main p1 ou p2
    switch(programmeLance)
    {
    //si le main est lancé
    case 0:
        for(int i = 0; i < m_nbAction; i++)
        {
            //comportement différent selon l'action
            Etat temp;
            temp = main->getActions().at(i)->getEtatDeLaction();
            switch(temp)
            {
            case Etat::AVANCER:
                std::cout << "AVANCER" << std::endl;
                r->avancer();
                break;
            case Etat::SAUTER:
                std::cout << "SAUTER" << std::endl;
                r->sauter();
                break;
            case Etat::TOURNERDROITE:
                std::cout << "TOURNERDROITE" << std::endl;
                r->tourner_droite();
                r->changerSprite();
                break;
            case Etat::TOURNERGAUCHE:
                std::cout << "TOURNERGAUCHE" << std::endl;
                r->tourner_gauche();
                r->changerSprite();
                break;
            case Etat::INTERAGIR:
                std::cout << "INTERAGIR" << std::endl;
                r->allumer();
                break;
            case Etat::P1:
                std::cout << "P1" << std::endl;

                p1->executer(r,main,p1,p2,1,fini,window);

                break;
            case Etat::P2:
                std::cout << "P2" << std::endl;
                p2->executer(r,main,p1,p2,2,fini,window);

                break;
            }
            r->afficher(window);
            window.display();
            sleep(0.75);
        }
        break;
```

```cpp
case Etat::P1:
    std::cout << "P1" << std::endl;
    baisseRep();
    if(MaxRep !=0)
    {
        p1->executer(r,main,p1,p2,1,fini,window);
    }
    break;
case Etat::P2:
    std::cout << "P2" << std::endl;
    baisseRep();
    if(MaxRep !=0)
    {
        p2->executer(r,main,p1,p2,2,fini,window);
    }
    break;
}
```

Paramètres : tous les conteneurs, un entier → programme lancé

Selon l'entier lance une séquence différentes

Récursivité gérée par baisseRep() et MaxRep

Ce qu’il manque au projet

Des objectifs n’ont pas été atteints :

- ❏ “Drag and Drop” (permet aussi d’ajouter en plein milieu d’un conteneur) : problème de conception !
- ❏ Les animations de déplacement
- ❏ Charte graphique précise
- ❏ Système de score
- ❏ Prévisualisation dans l’éditeur

Améliorations possibles

- Meilleure conception aurait pu être envisagée (ou bien respecter celle prévue)
  - Robot pointe sur une case
  - Classes héritants d'Actions et non pas un type
- Passer plus de temps sur les destructeurs (et pointeurs en général)
  - Beaucoup d'erreurs sur les pointeurs
- "Drag and Drop"
- Nettoyer les classes inutiles
  - Niveau
  - Fusionner CaseEditeur et Cases
- Position du robot : pointeur de Case et non pas en brute

Capitalisation d’expérience

- ❏ Apprentissage d’une meilleure gestion de projet
  - ❏ Gestion du temps
  - ❏ Gestion des tâches
- ❏ Utilisation d’outils de partage de source et Git
- ❏ Travail d’équipe

Conclusion

Objectifs initiaux réalisés

+ etages
+ 2D