Université de Namur

Faculté d'Informatique

Projet de programmation INFOB236

Rapport de modélisation en Processing

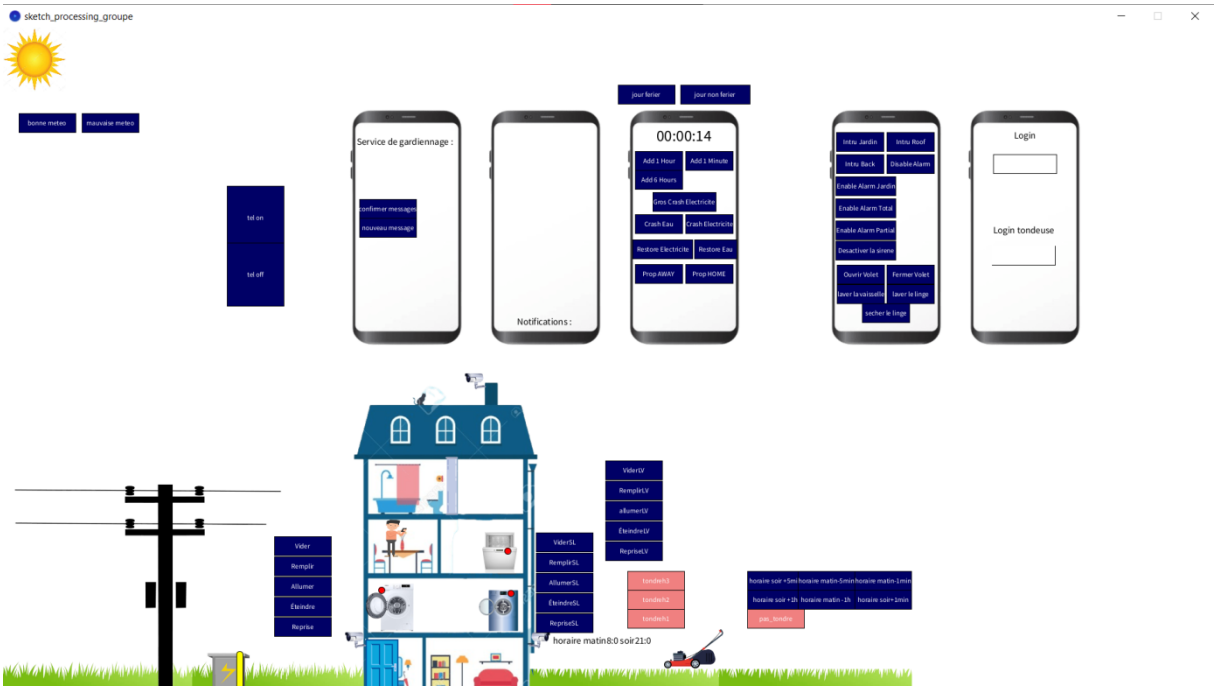
Auteur :
NILS DE BRUYCKER
AMER HAMDACHE
SYLIA SAD
ABDERRAHIM SEDDIKI

Professeur :
JEAN-MARIE JACQUET
Assistante :
MANEL BARKALLAH

2023-2024

Rapport Processing :

Image interface :



Voici l'interface de l'animation processing, quand on arrive on a le téléphone qui est éteint, il faut l'allumer en cliquant sur le bouton **« telOn »**,puis **choisir entre bonne et mauvaise météo, jour férié ou non férié** avec les boutons qu'ils lui sont consacrés (ces boutons ont été rajouté vu que le fonctionnement de certaines machines varie selon la météo et selon le jour ,s'il est férié ou pas).

Ensuite,il faut **faire le login** en appuyant sur la zone Login , un message vous demandons d'entrer le mot de passe sera affiché , une fois le mot de passe entré (**il faut entré le mot de passe suivant :1234**), on peut appuyer sur le bouton « **Confirm** » pour faire disparaitre le message et ceci , est valable pour toutes les autres notifications. Une fois le login fait, on pourra **faire marcher les machines, (veillez à les remplir avant lancement via le bouton « «Remplir »+ « abréviation du nom de la machine, et appuyer aussi sur les boutons laver/sécher le linge laver la vaisselle dans le téléphone puis après appuyer sur le bouton allumer qui est à côté de ces machines »)** chacune avec les boutons qu'il lui faut, de même pour activer/désactiver l'alarme selon le mode qu'on souhaite ; si la sirène est déclenchée on peut la désactiver mais avant il faut appuyer sur le bouton « PropAtHome » (c'est-à-dire le propriétaire est à proximité). **Pour la tondeuse il faut d'abord faire le Login** dans l'espace réservé(le **mot de passe pour la tondeuse est : 4321**) .Les messages envoyés au **service de gardiennage** apparaitront dans le téléphone nommé Service de gardiennage.

Veuillez noter qu'on ne peut pas ouvrir les volets si l'alarme est en mode total.

Implémentation :

Le fichier `**sketch_processing_groupe.pde**` est le script principal de notre projet en Processing, Le script implémente plusieurs fonctions pour interagir avec divers appareils et services à travers une interface utilisateur graphique. Voici un détail approfondi des méthodes et des structures utilisées dans notre script :

Importations
- processing.core.PApplet et processing.core.PImage : Ces importations permettent d'utiliser les fonctions de base de Processing pour l'affichage graphique et la manipulation des images.

Variables Globales
- Diverses instances de classes telles que `MachineALaver`, `SecheLinge`, `LaveVaisselle`, `Bouton`, et autres sont déclarées pour gérer les différents éléments de l'interface utilisateur et les appareils de la maison.
- boolean phone_on : Indique si le téléphone est actif ou non.
- ArrayList<String> notifications : Stocke les notifications à afficher à l'utilisateur.
- boolean showNotification : Un booléen pour contrôler l'affichage des notifications.

Méthodes Principales
1. setup() :
   - Initialisation de la fenêtre et chargement du fond d'écran.
   - Création des instances pour les différents appareils et services (comme la tondeuse, l'alarme, etc.).
   - Initialisation des boutons qui permettent de contrôler ces appareils et services.

2. draw():
   - Affichage de l'arrière-plan et mise à jour de l'état de tous les appareils et services en temps réel.
   - Gère l'affichage des notifications et des éléments interactifs en fonction de l'état des variables (comme `phone_on`).

3. mousePressed():
   - Gère les interactions avec les boutons. Chaque bouton, lorsqu'il est pressé, peut déclencher une action spécifique comme allumer ou éteindre un appareil, ouvrir ou fermer les volets, activer ou désactiver l'alarme, etc.
   - Ajoute des notifications à la liste basée sur les actions effectuées.

4. mouseReleased() :
   - Reset les états des boutons après que la souris est relâchée.

5. drawTime() :
   - Affiche l'heure actuelle sur l'interface utilisateur.

6. drawNotifications():

- Affiche les notifications accumulées dans la liste sur l'interface utilisateur.

Fonctionnalités Clés
- Gestion des appareils : Contrôle de divers appareils comme la machine à laver, le sèche-linge, et le lave-vaisselle avec des boutons pour les activer, les remplir, les vider, etc.
- Sécurité: Gestion de l'alarme avec la possibilité d'activer/désactiver ou simuler des intrusions.
- Gestion des volets: Ouvrir et fermer les volets via des boutons.
- Gestion de la tondeuse : Contrôle de la tondeuse, y compris la programmation de son horaire de fonctionnement.
- Notifications : Système de notifications pour informer l'utilisateur des actions réalisées ou des alertes.
- Login : Fonctionnalités de login pour sécuriser l'accès à certaines opérations.

Le fichier **`login.pde`** implémente une classe `Login` pour gérer l'authentification dans notre application. Cette classe utilise une interface utilisateur simple pour saisir un mot de passe et vérifier si celui-ci correspond à un mot de passe prédéfini qui est « 1234 ». Voici les détails de cette classe et de ses méthodes :

Classe Login

Attributs
- boolean login : Indique si l'utilisateur est connecté.
- String password : Le mot de passe attendu pour la connexion.
- String user_input: Stocke la saisie de l'utilisateur.
- int X, int Y : Coordonnées pour positionner le champ de saisie dans l'interface.
- boolean ecrire : Un booléen qui contrôle si l'application doit accepter la saisie de l'utilisateur.

Constructeur
- **Login(int x, int y, String Password)** : Initialise les attributs avec les valeurs fournies. L'utilisateur n'est pas connecté par défaut (`login = false`), et le champ de saisie est initialement vide.

Méthodes
1. void enter_password() :
   - Vérifie si le texte saisi par l'utilisateur correspond au mot de passe attendu.
   - Si oui, `login` est mis à `true`, indiquant que l'utilisateur est connecté, et la saisie est désactivée (`ecrire = false`).

2. void display() :
   - Affiche un champ de saisie de texte sur l'interface.
   - Le champ est affiché seulement si l'utilisateur n'est pas connecté (`login == false`).
   - Un rectangle blanc avec du texte noir montre où l'utilisateur peut saisir le mot de passe.

3. void get_input_from_user() :
   - Gère la saisie de l'utilisateur.
   - Si `ecrire` est vrai et une touche est pressée, cette méthode gère l'entrée.
   - Si l'utilisateur appuie sur la touche BACKSPACE, le dernier caractère du `user_input` est supprimé.

- Si l'utilisateur appuie sur ENTER ou RETURN, la méthode `enter_password()` est appelée pour vérifier le mot de passe, puis `user_input` est vidé et `ecrire` est mis à `false`.
- Pour toute autre touche, le caractère correspondant est ajouté à `user_input`.

Fonctionnalités Clés
- Gestion des sessions : La classe permet de vérifier si un utilisateur peut se connecter avec un mot de passe défini.
- Interaction utilisateur : Offre une interface textuelle simple pour la saisie du mot de passe.
- Sécurité : Empêche l'accès à certaines fonctionnalités tant que l'utilisateur n'est pas authentifié.

Le fichier `**Bouton.pde**` définit une classe `Bouton` pour créer et gérer des boutons interactifs dans une interface graphique. Cette classe permet de dessiner des boutons, de détecter les interactions avec la souris et de gérer les actions lorsqu'ils sont cliqués. Voici les détails des différentes composantes de la classe `Bouton` :

Classe Bouton

Attributs
- int cx, cy : Coordonnées du centre du bouton.
- int largeur, hauteur : Dimensions du bouton.
- String blabel : Texte affiché sur le bouton.
- color couleur : Couleur principale du bouton.
- color c_nonClickable, c_Clickable, c_default : Couleurs pour différents états du bouton (non cliquable, cliquable, par défaut).
- boolean dessus : Indique si la souris est au-dessus du bouton.
- boolean clique : Indique si le bouton a été cliqué.

Constructeur
- Bouton(int xp, int yp, int lgp, int htp, String bl): Initialise les attributs avec les valeurs fournies. La couleur par défaut est définie à une teinte de bleu.

Méthodes
1. void maj_souris() :
   - Met à jour l'attribut `dessus` en vérifiant si la souris est au-dessus du bouton. Cette vérification prend en compte les coordonnées de la souris et les dimensions du bouton.

2. boolean selectionne() :
   - Vérifie si le bouton est sélectionné (c'est-à-dire si la souris est dessus et un clic a été effectué).
   - Déclenche des actions spécifiques si le bouton est cliqué, comme l'ajout de temps à une horloge ou à un compteur (ces actions dépendent du texte du bouton).
   - Renvoie `true` si le bouton est cliqué, sinon `false`.

3. void majcouleur(boolean s) :
   - Met à jour la couleur du bouton en fonction de l'état passé en paramètre (`s`). Si `s` est vrai, le bouton devient "cliquable" (vert), sinon il prend la couleur "non cliquable" (rouge pâle).

4. void relache() :
   - Réinitialise l'attribut `clique` à `false`, indiquant que le bouton n'est plus en état de clic.

5. void affiche() :
   - Gère l'affichage du bouton. Cette méthode appelle `maj_souris()` pour mettre à jour la position de la souris.
   - Dessine le rectangle du bouton en utilisant les attributs de style et de couleur appropriés, en tenant compte de l'état `dessus` pour changer la couleur de remplissage lorsque la souris survole le bouton.
   - Affiche le texte du bouton centré dans le rectangle.

Fonctionnalités Clés
- Interactivité : La classe permet de créer des boutons qui réagissent aux actions de la souris, facilitant la création d'une interface utilisateur interactive.
- Flexibilité : Les boutons peuvent exécuter des actions spécifiques basées sur leur label, rendant la classe adaptable à différents contextes (par exemple, ajouter du temps à un compteur).
- Visuel : Les boutons changent de couleur selon leur état, améliorant l'expérience utilisateur en fournissant un feedback visuel immédiat.

Le fichier `A**larme.pde`** définit une classe `alarm` destinée à gérer les différents états et configurations d'un système d'alarme pour notre maison. Voici une analyse détaillée de cette classe :

Classe alarm

Attributs
- boolean jardin, partial, total : Ces booléens représentent les différents états de l'alarme (jardin uniquement, partiel, total).
- PImage coneG, coneD : Images pour représenter visuellement l'activation de l'alarme à différents endroits.
- int x_back, y_back, x_front, y_front, x_roof, y_roof : Coordonnées utilisées pour positionner les icônes des alarmes dans l'interface utilisateur.

Constructeur
- alarm() : Initialise les états de l'alarme à `false` et charge les images des cônes utilisées pour l'affichage. Les coordonnées des images sont également initialisées à zéro.

Méthodes
1. void activateJardin():
   - Active l'alarme pour le jardin uniquement. Cela désactive les autres alarmes (partielle et totale) et définit les coordonnées pour afficher l'icône correspondante au jardin.

2. void activatePartial():
   - Active l'alarme partielle, qui couvre plus que le jardin mais n'est pas aussi étendue que l'alarme totale. Ajuste les coordonnées pour afficher les icônes à l'avant et à l'arrière.

3. void activateTotal() :
   - Active l'alarme totale, ce qui signifie que tous les secteurs sont surveillés (jardin, arrière, toit, etc.). Positionne les icônes pour une couverture totale.

4. void turnOff() :
   - Désactive toutes les formes de l'alarme et réinitialise les coordonnées des icônes à zéro.

5. boolean isJardin(), isPartial(), isTotal() :
   - Ces méthodes accesseurs retournent l'état actuel de l'alarme pour chaque zone (jardin, partiel, total).

6. void display() :
   - Affiche les icônes de l'alarme en fonction des états actifs. Utilise les coordonnées définies pour positionner correctement les icônes dans l'interface utilisateur.
   - Gère l'affichage conditionnel basé sur les états de l'alarme, en s'assurant que les icônes correctes sont affichées pour les zones activées.

Fonctionnalités Clés
- Flexibilité : La classe permet une gestion flexible des différentes zones d'alarme, ce qui est essentiel pour un système de sécurité domestique où différentes zones peuvent nécessiter une surveillance variable.
- Visuel : Les icônes sont utilisées pour donner un retour visuel immédiat sur les zones actuellement surveillées, améliorant ainsi l'interface utilisateur.
- Simplicité : Les méthodes sont conçues pour être simples et directes, facilitant l'intégration et la maintenance du code.

Le fichier `**intrusion.pde**` contient la définition de la classe `intrusion` qui gère la détection d'intrus et le contrôle des alarmes dans un système de sécurité domestique. Voici une explication détaillée de cette classe et de ses méthodes :

Classe intrusion

Attributs
- boolean intrusion, desactiv_sirene, jard, back, roof : Variables booléennes pour contrôler l'état de l'intrusion et la désactivation de la sirène, ainsi que pour identifier où l'intrusion a lieu (jardin, arrière, toit).
- PImage intrus, sirene : Images utilisées pour représenter visuellement un intrus et une sirène activée.
- int x_intrus, y_intrus, x_sirene, y_sirene : Coordonnées pour positionner les images des intrus et de la sirène.
- boolean intrudetect : Indique si une intrusion a été récemment détectée.
- proprio propri : Instance de la classe `proprio` utilisée pour vérifier si le propriétaire est présent.

Constructeur
- intrusion() : Initialise les états à `false`, charge les images nécessaires, et définit les positions initiales pour les images.

Méthodes
1. void intru_Jardin_detect(), void intru_Back_detect(), void intru_Roof_detect() :
   - Ces méthodes définissent les coordonnées de l'intrus en fonction de la zone où l'intrusion est détectée. Elles mettent à jour l'état de l'intrusion et réinitialisent les détecteurs pour d'autres zones.

2. void desactiv_siren(), void reactiv_siren() :
  - Ces méthodes contrôlent l'activation et la désactivation de la sirène.

3. void display() :
  - Gère l'affichage des intrus et des sirènes en fonction des zones où des intrusions ont été détectées.
  - Affiche des notifications si de nouvelles intrusions sont détectées, et ces notifications sont gérées par une autre instance ou module qui maintient un registre des messages ou des alertes.

4. boolean guard_desactiv_siren() :
- Retourne un booléen indiquant si le propriétaire est présent, ce qui peut affecter la possibilité de désactiver la sirène.

Fonctionnalités Clés
- Gestion Multi-Zone : La classe gère les intrusions dans différentes zones (jardin, arrière, toit) et ajuste l'affichage et les alertes en conséquence.
- Contrôle de la Sirène : Permet d'activer ou de désactiver la sirène basée sur l'état de l'intrusion et la présence du propriétaire.
- Interaction avec les Propriétaires : Utilise une instance de `proprio` pour déterminer si certaines actions (comme la désactivation de la sirène) sont autorisées.
- Feedback Visuel et Auditif : Utilise des images pour indiquer visuellement la présence d'un intrus et une sirène pour le feedback auditif.

Le fichier **`LaveVaisselle.pde`** décrit une classe `LaveVaisselle` qui gère les fonctionnalités d'un lave-vaisselle dans un environnement de maison intelligente. Cette classe s'occupe des opérations de base comme démarrer et arrêter le lave-vaisselle, ainsi que des fonctionnalités supplémentaires telles que la pause et la reprise des cycles. Voici un examen détaillé de la classe et de ses méthodes :

Classe LaveVaisselle

Attributs
- PImage imageVideLV, imagePleinLV : Images pour représenter le lave-vaisselle vide et plein.
- boolean estPleinLV, estAllumeLV, commande, estEnPauseLV : Booléens qui indiquent respectivement si le lave-vaisselle est plein, allumé, s'il y a une commande en cours, et s'il est en pause.
- int tempsRestantLV : Temps restant pour le cycle en cours, en minutes.
- int xLV, yLV : Coordonnées pour afficher l'image du lave-vaisselle.
- Temp heureFinirLV : Un objet de type Temp qui stocke le moment où le cycle actuel doit se terminer.

Constructeur
- LaveVaisselle(String vide, String plein, int x, int y) : Initialise le lave-vaisselle avec les images spécifiées et positionne l'appareil à l'emplacement donné. L'objet `heureFinirLV` est également initialisé pour gérer le temps.

Méthodes

1. void afficherLV() :
   - Affiche l'image du lave-vaisselle selon son état (plein ou vide) et montre visuellement s'il est allumé avec un indicateur lumineux vert, ou éteint avec un indicateur rouge.

2. void togglePleinLV() :
   - Change l'état de remplissage du lave-vaisselle (plein/vide).

3. void toggleAllumeLV():
   - Bascule l'état d'activation du lave-vaisselle. Si allumé, il démarre le compteur de fin de cycle. Si éteint, il réinitialise le compteur.

4. void pauseLV() :
   - Met le lave-vaisselle en pause, sauvegardant le temps restant avant la fin du cycle.

5. void repriseLV() :
   - Reprend le cycle là où il s'était arrêté après une pause.

6. boolean guard_toggle_pleinLV() :
   - Garde conditionnelle qui empêche de changer l'état plein/vide du lave-vaisselle s'il est allumé.

7. boolean guard_toggleLV() :
   - Conditions sous lesquelles le lave-vaisselle peut être allumé ou non, en tenant compte de restrictions horaires, de jours spécifiques, et de la disponibilité des ressources nécessaires comme l'eau et l'électricité.

8. boolean guard_toggle_eteintLV() :
   - Vérifie si le lave-vaisselle peut être éteint en fonction de la progression du cycle.

9. boolean guard_repriseLV() :
   - Conditions sous lesquelles le cycle du lave-vaisselle peut être repris après une pause.

Fonctionnalités Clés
- Gestion de l'État : La classe gère efficacement les différents états du lave-vaisselle (allumé, éteint, en pause, plein, vide) en fournissant une interface utilisateur intuitive.
- Feedback Visuel : Les méthodes d'affichage fournissent un retour visuel clair sur l'état de l'appareil, ce qui est crucial dans une interface de maison intelligente.
- Gestion du Temps : La gestion du temps avec des restrictions basées sur différents paramètres assure une utilisation optimale de l'appareil en fonction des besoins de l'utilisateur et des disponibilités en ressources.

Le fichier **`machineALaver.pde`** définit une classe `MachineALaver` pour contrôler les fonctionnalités d'une machine à laver dans un système de gestion domotique. La classe gère des actions telles que démarrer et arrêter la machine, la mettre en pause, et reprendre son fonctionnement. Voici une description détaillée de cette classe :

Classe MachineALaver

Attributs
- PImage imageVide, imagePlein : Images représentant la machine à laver vide et pleine.

- boolean estPlein, estAllume, commande, estEnPause : Booléens pour indiquer si la machine est pleine, allumée, si une commande est en cours, et si elle est en pause.
- int tempsRestant : Temps restant pour le cycle en cours, en minutes.
- int x, y : Coordonnées pour l'affichage de la machine à laver.
- Temp heureFinir : Un objet `Temp` pour gérer le temps de fin de cycle.

Constructeur
- MachineALaver(String vide, String plein, int x, int y) : Charge les images et initialise les coordonnées de la machine à laver. Un nouvel objet `Temp` est créé pour suivre la fin du cycle.

Méthodes
1. void afficher() :
   - Affiche l'image de la machine à laver en fonction de son état (plein ou vide) et montre un indicateur lumineux pour son état allumé ou éteint.

2. void togglePlein() :
   - Alterne l'état de la machine entre plein et vide.

3. void toggleAllume() :
   - Active ou désactive la machine. En activant, elle démarre le cycle et calcule l'heure de fin. En désactivant, elle réinitialise l'heure de fin et le temps restant.

4. void pause():
   - Met la machine en pause, sauvegardant le temps restant pour reprendre plus tard.

5. void reprise() :
   - Reprend le cycle là où il s'était arrêté, ajustant l'heure de fin en fonction du temps restant.

6. boolean guard_toggle_plein() :
   - Contrôle si l'état plein de la machine peut être modifié (ne peut pas être changé si la machine est allumée).

7. boolean guard_toggle() :
   - Vérifie si la machine peut être démarrée ou arrêtée, basé sur des conditions spécifiques comme l'heure, les jours spéciaux, l'état de la machine, et si les ressources nécessaires sont disponibles.

8. boolean guard_toggle_eteint() :
   - Détermine si la machine peut être éteinte, basé sur l'avancement du cycle.

9. boolean guard_reprise() :
   - Vérifie si la machine peut reprendre son cycle après une pause, en s'assurant que toutes les conditions requises sont remplies.

Fonctionnalités Clés
- Contrôle de l'État : La classe offre une gestion précise de l'état de la machine, y compris son remplissage, son activation, et son fonctionnement.
- Gestion du Temps : Elle calcule et ajuste le temps nécessaire pour compléter un cycle, ce qui est crucial pour planifier et optimiser l'utilisation de la machine.

- Feedback Visuel : Les indicateurs visuels renforcent l'interaction utilisateur en fournissant un retour immédiat sur l'état de la machine.

Le fichier **`SecheLinge.pde`** définit une classe `SecheLinge` qui gère les opérations d'un sèche-linge dans un système de maison intelligente. Cette classe inclut des fonctions pour démarrer et arrêter le sèche-linge, le mettre en pause et reprendre son fonctionnement, tout en contrôlant son état (plein ou vide). Voici un aperçu détaillé des composants et des fonctionnalités de la classe :

Classe SecheLinge

Attributs
- PImage secheLingeVide, secheLingePlein : Images représentant le sèche-linge lorsqu'il est vide et plein.
- boolean secheLingeEstPlein, secheLingeEstAllume, commande, estEnPauseSL : Indicateurs de l'état du sèche-linge (plein, allumé, avec une commande active, et en pause).
- int tempsRestantSL: Temps restant pour le cycle actuel, en minutes.
- int posXSL, posYSL: Coordonnées pour le placement du sèche-linge dans l'interface utilisateur.
- Temp heureFinirSL : Temps calculé pour la fin du cycle en cours.

Constructeur
- SecheLinge(String vide, String plein, int x, int y) : Charge les images pour les états plein et vide du sèche-linge et initialise les coordonnées de placement. Crée également un objet `Temp` pour gérer le temps de fin de cycle.

Méthodes
1. void afficherSL() :
   - Affiche l'image appropriée du sèche-linge selon son état (plein ou vide) et montre un indicateur lumineux vert ou rouge pour signaler si l'appareil est allumé ou éteint.

2. void togglePleinSL() :
   - Bascule l'état de remplissage du sèche-linge (plein ou vide).

3. void toggleAllumeSL() :
   - Active ou désactive le sèche-linge, démarrant ou arrêtant le cycle. Si allumé, initialise le compteur de fin de cycle ; si éteint, réinitialise ce compteur et le temps restant.

4. void pauseSL() :
   - Met le sèche-linge en pause, sauvegardant le temps restant avant la fin du cycle.

5. void repriseSL() :
   - Reprend le cycle du sèche-linge là où il s'était arrêté après une pause, ajustant l'heure de fin en fonction du temps restant.

6. boolean guard_toggle_pleinSL() :
   - Contrôle si l'état plein/vide du sèche-linge peut être modifié, ce qui n'est pas possible si l'appareil est allumé.

7. boolean guard_toggleSL() :
   - Vérifie si le sèche-linge peut être démarré ou arrêté, basé sur plusieurs conditions telles que l'heure, la présence du propriétaire, et l'accès à l'électricité et à l'eau.

8. boolean guard_toggle_eteintSL() :
   - Détermine si le sèche-linge peut être éteint, en fonction de l'avancement du cycle.

9. boolean guard_repriseSL() :
   - Conditions sous lesquelles le cycle du sèche-linge peut être repris après une pause.

Fonctionnalités Clés
- Gestion des États : Permet une manipulation précise des différents états du sèche-linge, améliorant la flexibilité et l'efficacité de son utilisation.
- Feedback Visuel : Fournit des indicateurs visuels clairs pour l'état actuel du sèche-linge, essentiels pour une interface utilisateur intuitive.
- Gestion du Temps : Calcule et ajuste le temps nécessaire pour terminer un cycle, ce qui est crucial pour planifier l'utilisation de l'appareil.

Le fichier `**messages.pde**` définit une classe `Messages` qui est conçue pour gérer les notifications de sécurité dans un système de maison intelligente, en particulier en ce qui concerne les intrusions. La classe enregistre et rafraîchit les messages d'intrusion, en intégrant des données temporelles pour suivre les événements. Voici une analyse détaillée de cette classe :

Classe Messages

Attributs
- ArrayList<String> messages : Liste contenant les messages d'alerte.
- Temp date_derniere_intru_detecte: Enregistre la date et l'heure de la dernière intrusion détectée.
- String type_derniere_intru : Spécifie le type de la dernière intrusion détectée (par exemple, jardin, toit, etc.).
- Temp date_pour_refresh : Gère le moment pour rafraîchir les messages.
- boolean recu : Indique si un message a été reçu et non encore traité.

Constructeur
- Messages() : Initialise les attributs, créant des objets pour les dates et une liste pour les messages.

Méthodes
1. void register_new_intrusion() :
   - Enregistre une nouvelle intrusion en ajoutant un message détaillé à la liste `messages`. La date et le type de l'intrusion sont mis à jour.
   - La première détection d'intrusion initialise également `date_pour_refresh` si elle n'a pas déjà été définie.

2. boolean guard_refresh() :
   - Contrôle si les messages doivent être rafraîchis, en fonction de l'écart de temps depuis la dernière détection et si le téléphone est activé. La condition s'assure que 5 minutes se sont écoulées depuis la dernière intrusion détectée.

3. void refresh() :
  - Rafraîchit la liste des messages en fonction du temps écoulé depuis le dernier rafraîchissement. Pour chaque intervalle de 5 minutes, un nouveau message est généré, indiquant le type et la date de l'intrusion.
  - Met à jour `date_pour_refresh` pour le moment actuel, et marque les messages comme reçus si le téléphone est activé.

Fonctionnalités Clés
- Gestion Temporelle des Alertes : La classe utilise des objets `Temp` pour contrôler le moment des intrusions et des rafraîchissements, garantissant que les notifications sont générées et gérées de manière opportune.
- Type d'Intrusion : Spécifie le type d'intrusion pour des notifications précises et des actions de suivi appropriées.
- Rafraîchissement Dynamique : Les messages sont régulièrement mis à jour pour refléter les activités récentes, aidant à maintenir une surveillance continue et à réagir rapidement aux menaces.

Le fichier **`Eau_et_electricite.pde`** définit une classe `Eau_et_electricite` qui gère les infrastructures et les interruptions des services d'eau et d'électricité. Cette classe joue un rôle crucial dans la gestion des ressources vitales et dans la réaction à des situations d'urgence comme les pannes de courant ou de l'eau. Voici une analyse détaillée de cette classe :

Classe Eau_et_electricite

Attributs
- boolean eau, electricite: Indiquent si l'eau et l'électricité sont disponibles.
- Temp date_crash_electricite: Enregistre le moment d'une panne d'électricité.
- boolean bare_energie: Indique l'état de l'alimentation énergétique.
- PImage image_eau, image_electricite, image_compteur : Images représentant l'état visuel des infrastructures d'eau, d'électricité, et du compteur.

Constructeur
- Eau_et_electricite(): Initialise les systèmes avec l'eau et l'électricité disponibles et charge les images appropriées pour les états par défaut et en panne.

Méthodes
1. void crash_eau() :
  - Simule une panne d'eau, mettant à jour l'image et interrompant les appareils dépendants de l'eau.

2. void crash_electricite():
  - Simule une panne d'électricité, mettant à jour l'image et notifiant les risques pour les appareils en fonction.

3. void gros_crash_electricite():

- Gère un grand crash électrique, désactivant tous les appareils et systèmes qui dépendent de l'électricité, y compris l'alarme et la tondeuse.

4. void restore_eau():
- Restaure le service d'eau, mettant à jour l'image pour refléter la résolution du problème.

5. void restore_electricite() :
- Restaure le service d'électricité, mettant à jour l'image et rétablissant la barre d'énergie.

6. boolean guard_big_crash():
- Vérifie si un grand crash électrique peut être déclaré, basé sur le temps écoulé depuis la dernière panne enregistrée.

7. void display() :
- Affiche les images actuelles pour l'eau, l'électricité et le compteur d'énergie, offrant une visualisation de l'état actuel des ressources dans la maison.

Fonctionnalités Clés
- Gestion des Ressources Vitales : La classe assure la surveillance et la gestion des services essentiels de l'eau et de l'électricité, jouant un rôle central dans la fonctionnalité globale de la maison intelligente.
- Réaction aux Pannes: Elle permet de réagir rapidement aux pannes, minimisant les perturbations des appareils domestiques et maintenant la sécurité des occupants.
- Feedback Visuel: Fournit des indicateurs visuels immédiats de l'état des ressources, facilitant la compréhension de la situation par les utilisateurs.

Le fichier `**Proprio.pde**` définit une classe `proprio` pour gérer la présence du propriétaire dans ca maison intelligente. Cette classe est cruciale pour ajuster les comportements des différents systèmes de la maison en fonction de la présence ou de l'absence du propriétaire. Voici un détail sur la structure et les fonctionnalités de cette classe :

Classe proprio

Attributs
- boolean is_near : Indique si le propriétaire est proche (dans la maison) ou non.
- PImage impro : Image représentant le propriétaire.
- int x, y : Coordonnées pour l'affichage de l'image du propriétaire.

Constructeur
- proprio() : Initialise le propriétaire comme étant proche et charge une image représentative. Définit aussi les coordonnées initiales pour l'affichage.

Méthodes
1. void go_away():
- Change l'état du propriétaire à "absent" et met à jour les coordonnées pour que l'image du propriétaire soit placée hors de la vue principale, symbolisant son absence de la maison.

2. void go_home():

- Rétablit l'état du propriétaire à "présent" et remet les coordonnées à leur position initiale, montrant que le propriétaire est de retour à la maison.

3. boolean is_here() :
- Retourne l'état actuel du propriétaire, indiquant s'il est près ou non.

4. void display():
- Affiche l'image du propriétaire à l'écran en fonction de son état. Si le propriétaire est présent, l'image est affichée à sa taille normale et aux coordonnées spécifiées. Si absent, l'image peut être réduite ou affichée différemment pour représenter visuellement cette absence.

Fonctionnalités Clés
- Gestion de Présence : La classe offre un moyen simple mais efficace de gérer et de visualiser la présence du propriétaire dans la maison.
- Interaction avec D'autres Systèmes : L'état de présence du propriétaire peut être utilisé pour ajuster le comportement d'autres systèmes dans la maison, comme la sécurité, l'éclairage, et le chauffage.
- Feedback Visuel: Fournit une représentation visuelle de la présence du propriétaire, améliorant l'interface utilisateur et enrichissant l'interaction avec le système.

Le fichier `**Temp.pde**` décrit une classe `Temp` conçue pour gérer le temps dans notre système, incluant des fonctions pour avancer le temps, afficher l'heure actuelle, et gérer les jours fériés. Cette classe joue un rôle central dans la synchronisation et le contrôle des tâches automatisées basées sur le temps dans la maison. Voici une explication détaillée de cette classe :

Classe Temp

Attributs
- int jour, heure, minute, compteur : Variables pour stocker l'heure actuelle et un compteur pour des opérations temporelles.
- boolean ferie: Indique si le jour actuel est un jour férié, influençant certaines opérations ou comportements dans la maison.

Constructeur
- Temp() : Initialise les variables de temps à zéro et définit `ferie` à `false`.

Méthodes
1. void forward(int jour_en_plus, int heure_en_plus, int minute_en_plus) :
- Avance le temps en ajoutant des jours, des heures et des minutes spécifiées aux valeurs actuelles. Cette méthode ajuste également les heures et les minutes pour s'assurer qu'elles restent dans les limites valides (heures < 24, minutes < 60).
- Déclenche un rafraîchissement des messages si nécessaire après l'avancement du temps.

2. void slow_forward():
- Avance le temps lentement, en incrémentant les minutes par étape. Ce mécanisme est utilisé pour simuler le passage du temps de manière plus contrôlée.

3. void display():
   - Affiche l'heure actuelle sous un format spécifique. Si c'est un jour férié, le texte correspondant est également affiché.

4. void ferie_on(), void ferie_off():
   - Activer ou désactiver le statut de jour férié, influençant les opérations qui peuvent dépendre de cette information.

5. void maj_bouton():
   - Met à jour l'état de divers boutons dans l'interface utilisateur, reflétant les changements dans le système et réagissant à des conditions telles que la présence du propriétaire ou des jours spécifiques.

Fonctionnalités Clés
- Gestion du Temps : Fournit un contrôle précis sur le temps, crucial pour planifier et exécuter des tâches automatisées dans une maison intelligente.
- Support des Jours Fériés : Permet d'ajuster les comportements ou les opérations basées sur le statut de jour férié, ce qui peut affecter tout, des horaires de l'éclairage aux notifications de sécurité.
- Intégration avec Autres Systèmes : La méthode `maj_bouton` indique que la classe `Temp` est étroitement intégrée avec d'autres composants du système, utilisant le temps pour activer ou désactiver des fonctionnalités à travers la maison.

Le fichier **`volets.pde`** définit une classe `volet` qui gère le fonctionnement des volets dans le système. Cette classe permet d'ouvrir et de fermer les volets, en offrant une gestion visuelle de leur état. Voici une explication détaillée de cette classe :

Classe volet

Attributs
- boolean ouvert : Indique si les volets sont ouverts ou fermés.
- int x, y : Coordonnées pour positionner les images des volets lorsqu'ils sont fermés.
- PImage imvolet : Image représentant les volets.

Constructeur
- volet() : Initialise les volets comme ouverts par défaut, définit les coordonnées initiales, et charge l'image des volets.

Méthodes
1. void ouvrir_volet() :
   - Ouvre les volets en mettant à jour l'état à `true`.

2. void fermer_volet() :
   - Ferme les volets en mettant à jour l'état à `false` et ajuste les coordonnées pour l'affichage.

3. void display() :
   - Affiche les images des volets en fonction de leur état. Si les volets sont fermés, plusieurs images sont affichées côte à côte pour créer l'effet de plusieurs panneaux de volets.

4. boolean guard_volet() :
   - Détermine si les volets peuvent être manipulés, en se basant sur la condition que l'utilisateur doit être connecté et qu'il n'y a pas d'alarme totale activée.

Fonctionnalités Clés
- Gestion des États: Permet une manipulation simple des volets, en changeant leur état entre ouvert et fermé selon les besoins de l'utilisateur.
- Feedback Visuel : Fournit une représentation visuelle claire de l'état des volets, améliorant ainsi l'expérience utilisateur en montrant visuellement l'état ouvert ou fermé.
- Contrôle Conditionnel : Intègre des conditions de sécurité, telles que l'état de connexion de l'utilisateur et l'état de l'alarme, pour garantir que les actions sur les volets sont sécurisées et appropriées.

La classe `Tondeuse` dans le fichier **`tondeuse.pde`** est conçue pour gérer les opérations d'une tondeuse robotisée dans un système de maison intelligente, notamment le contrôle de son horaire de tonte, l'interaction avec les conditions météorologiques, et la sécurisation via un système de login. Voici un aperçu détaillé des composants et des fonctionnalités de cette classe :

Classe Tondeuse

Attributs
- int[][] horaire : Tableau stockant les heures de début et de fin de tonte pour la soirée et le matin.
- String mot_de_passe : Mot de passe pour l'accès sécurisé à la tondeuse.
- Temp heure_de_fin : Heure prévue pour la fin de la tonte.
- boolean on: Indique si la tondeuse est en fonctionnement.
- int rectangles_herbe_Y : Position Y pour le visuel de tonte dans l'interface graphique.
- PImage visuel, meteo_visuel: Images pour la tondeuse et les conditions météorologiques.
- int X, Y : Coordonnées de la tondeuse dans l'interface graphique.
- boolean sens : Direction actuelle de la tondeuse.
- Login login_tondeuse : Système de login pour la tondeuse.
- boolean meteo : État actuel du temps (beau ou pluvieux).
- int temp_restant_a_interuption : Temps restant avant la prochaine interruption programmée.

Méthodes
1. void set_horaire_soir_plus_1h(), set_horaire_soir_plus_5min(), etc. :
   - Ajuste les horaires programmés pour la tondeuse, permettant une flexibilité dans la planification des tontes.

2. void beau_temp(), void mauvais_temp() :
   - Ajuste les visuels en fonction des conditions météorologiques, influençant le comportement de la tondeuse.

3. boolean guard_tondre() :
   - Vérifie si les conditions sont réunies pour démarrer la tonte, incluant le login, la disponibilité électrique, et la météo favorable.

4. void tondre(int niveau_de_tonte) :

- Active la tondeuse en fonction de l'heure programmée et ajuste visuellement la hauteur de coupe sur l'interface.

5. void display() :
- Affiche la tondeuse et les indicateurs de tonte sur l'interface graphique.

6. void check() :
- Contrôle la position de la tondeuse et ajuste sa trajectoire en fonction des limites prédéfinies.

7. void auto() :
- Exécute automatiquement des fonctions telles que la reprise après pause ou l'arrêt, selon les conditions.

8. boolean guard_reprise() :
- Conditions requises pour reprendre la tonte après une interruption.

9. void reprise() :
- Reprise de la tonte après une interruption, recalculant l'heure de fin en fonction du temps restant.

10. boolean guard_off() :
- Conditions pour arrêter la tonte, généralement après la fin du cycle programmé.

11. void off() :
- Désactive la tondeuse et réinitialise les paramètres de fin.

Fonctionnalités Clés
- Gestion Flexible du Temps : Permet un contrôle précis sur les horaires de tonte, ajustables selon les préférences de l'utilisateur et les conditions extérieures.
- Interaction Météo : Adapte son fonctionnement aux conditions météorologiques, optimisant l'efficacité et la sécurité de la tonte.
- Sécurité et Contrôle d'Accès : Utilise un système de login pour sécuriser l'accès et le contrôle de la tondeuse.