

Programme ton robot en C

1/ L'ÉVEIL D'OKTAL



Jean-Michel Hautbois

L'Éveil d'Oktal

L'Éveil d'Oktal

Programme ton robot en C

Jean-Michel Hautbois



En application de l'art. L.137-2.-I. du code de la propriété intellectuelle, toute reproduction et/ou divulgation de parties de l'œuvre dépassant le volume prévu par la loi est expressément interdite.

L'analyse automatique du travail, pour obtenir des informations, notamment sur les motifs, les tendances et les corrélations ("data and text mining") est interdite.

© 2026, Jean-Michel Hautbois

Illustrations et couverture : Clément Berneron

Édition et impression :

BoD · [Books on Demand GmbH](#), Überseering 33, 22297 Hambourg, Allemagne

ISBN: 978-2-3227-8059-4

Dépôt légal : mai 2026

Loi n° 49-956 du 16 juillet 1949 sur les publications destinées à la jeunesse

<https://press.yoseli.org/>

Remerciements

À Claire, ma compagne de tant d'années. À côté de qui tout devient possible, même les projets entrepris en silence.

À mes parents, qui ont déposé entre mes mains le premier ordinateur et n'ont jamais pensé à me demander de le refermer. À mes frères, Guillaume et Flavien, et un merci particulier à Flavien pour son regard sur le manuscrit, précieux pour affiner le ton et le rythme.

À Léandre, Armand et Élise, mes trois étincelles. Une pensée particulière pour Léandre, mon tout premier relecteur et testeur, qui a traqué les passages trop compliqués avec la rigueur d'un ingénieur en herbe. Armand et Élise, promis, les prochains passeront entre vos mains. Et à toute la tribu qui gravite autour, neveux, nièces, beaux-parents, cousins, les autres : vous n'avez rien demandé, et c'est aussi pour vous que ce livre existe.

À Clément Berneron, qui a donné à Oktal ses yeux cyan et ce regard curieux qui traverse toutes les pages. Sans son trait, le robot serait resté une idée sur une feuille.

À John Maloney, co-créateur de Scratch et créateur de MicroBlocks, dont le travail a inspiré des générations de jeunes programmeurs. C'est en voyant des collégiens découvrir la joie de programmer avec des blocs colorés que l'idée de ce livre est née : leur offrir un pont vers le code réel.

Enfin, à toutes celles et ceux qui font vivre le logiciel libre et l'open source. Compilateurs, noyaux, RTOS, bibliothèques, outils de build : la plupart sont écrits par des gens qui partagent leur travail sans rien demander. Ce livre n'existerait pas sans Zephyr, sans la micro :bit Educational Foundation, sans le projet GNU, sans la grande famille Linux dans laquelle je navigue depuis plus de vingt ans, et sans mille briques anonymes que personne ne remercie jamais. Si tu lis ces pages, tu tiens le fruit de ce partage. Le libre, c'est ça : ouvrir le capot pour que d'autres puissent construire la suite.

Jean-Michel Hautbois – Acigné, mai 2026

À toi qui ouvres ce manuel

Ce document a été retrouvé dans les Archives de la Friche de Silicium, secteur 7-G, sous une pile de circuits imprimés.

Coordonnées d'origine : zone grise. Date d'archivage : tampon illisible. Époque : indéterminée.

La **Friche de Silicium**, c'était le Centre de Recherche en Robotique Autonome. On y fabriquait les **Unités OKT**, douze robots compacts numérotés de OKT-1 à OKT-12, conçus pour explorer et entretenir ce que les humains ne voulaient plus toucher.

Puis quelque chose s'est propagé dans les systèmes. On l'appelle **le Signal**. Les robots ont commencé à dysfonctionner, puis à s'éteindre un par un. Les humains ont évacué. Le centre a été scellé et oublié.

Le manuel que tu tiens entre les mains n'est pas neuf. Quelqu'un l'a possédé avant toi. Quelqu'un qui a pris le temps d'annoter les marges, de souligner des passages, de griffonner des avertissements.

Ces annotations portent un nom : les *marginalia*. Au Moyen Âge, les moines copistes y laissaient des notes dans les marges des manuscrits. Des messages d'un lecteur à l'autre, traversant les siècles.

Nous ne savons pas qui a laissé ces *marginalia*. Les Archives l'appellent simplement **l'Archiviste**. Ses notes sont parfois obscures, parfois salvatrices. Certaines mentionnent une mystérieuse **OKT-7** et des coordonnées étranges. D'autres contiennent des codes hexadécimaux que personne n'a encore déchiffrés.

Une chose est sûre : l'Archiviste connaissait ces robots mieux que personne. Il ou elle les a réparés, documentés, et a tenu assez longtemps pour laisser ces traces.

Lis ses notes. Certaines t'éviteront de faire les mêmes erreurs que lui.

— Les Archives de la Friche

Table des matières

Remerciements	5
Liste des figures	26
Liste des tableaux	28
Introduction	29
Prologue	35
I Les fondations	37
1 De la programmation visuelle au C	38
1.1 Rappel : la programmation par blocs	38
1.1.1 La structure d'un programme MakeCode	39
1.2 Le concept en C : du texte au lieu de blocs	40
1.2.1 La correspondance Blocs → C	40
1.2.2 Ligne par ligne	40
1.2.3 Et la boucle "toujours" ?	41
1.2.4 Les commentaires	42
1.3 Les règles de syntaxe	43
1.3.1 Les erreurs courantes	43
1.3.2 Les caractères spéciaux	43
1.4 Sur le robot : compiler et exécuter	44
1.4.1 Blocs vs C : le processus	44
1.4.2 Qu'est-ce que la compilation ?	44
1.4.3 Voir le résultat	45
1.5 Tester sans le robot	45
1.5.1 Le Playground OktalOS	46

1.6	Exercices	46
1.7	Mini-projet : Présentation du robot	48
1.8	Résumé	49
2	Dans les coulisses du robot	50
2.1	Le système d'exploitation (OS)	50
2.1.1	Que fait un OS ?	51
2.2	Et sur le robot ?	51
2.2.1	Pas d'OS sur le micro:bit ?	51
2.2.2	Zephyr : un mini-OS	51
2.3	Le processeur : le cerveau du robot	53
2.3.1	Que fait un processeur ?	53
2.3.2	L'ALU : la calculatrice	53
2.3.3	Les registres : mémoire ultra-rapide	54
2.3.4	Le langage du processeur : le binaire	55
2.4	La mémoire	55
2.5	Du code au binaire : la compilation	56
2.5.1	Le problème	56
2.5.2	La solution : le compilateur	56
2.5.3	Exemple concret	56
2.5.4	Schéma récapitulatif	58
2.6	Comment le processeur parle aux périphériques	59
2.6.1	Le problème : synchroniser l'échange	59
2.6.2	L'horloge : le métronome du bus	60
2.6.3	Exemple : allumer la LED avant gauche	60
2.6.4	GPIO, I2C, SPI : les bus de notre robot	61
2.7	Pourquoi apprendre le C ?	62
2.8	Résumé	62
2.9	Mini-projet : Explorateur de mémoire	63
2.10	Exercices	64
3	Variables et types	66
3.1	Rappel MakeCode : les variables	66
3.2	Le concept en C : déclarer une variable	67
3.2.1	La syntaxe de base	67
3.2.2	Comparaison MakeCode / C	67
3.2.3	Modifier une variable : affectation	67
3.3	Les types de données	67
3.3.1	Les types entiers	68

3.3.2	Le type booléen	68
3.4	Opérations sur les variables	69
3.4.1	Opérations arithmétiques	69
3.4.2	Raccourcis	70
3.4.3	Les constantes avec const	71
3.5	Sur le robot : contrôler la vitesse	71
3.5.1	Le code pour les moteurs	71
3.5.2	Afficher des variables avec printf	72
3.6	Attention au débordement!	73
3.7	Exercices	73
3.8	Mini-projet : Affichage de télémétrie	75
3.9	Pour aller plus loin	75
3.9.1	La mémoire	75
3.10	Résumé	76
4	Conditions et choix	77
4.1	Rappel MakeCode : si...alors	77
4.2	Le concept en C : if, else	78
4.2.1	La structure if	78
4.2.2	Ajouter une alternative : else	78
4.2.3	Plusieurs conditions : else if	79
4.3	Les opérateurs de comparaison	80
4.4	Combiner des conditions : opérateurs logiques	81
4.4.1	ET logique : &&	81
4.4.2	OU logique : 	81
4.4.3	NON logique : !	82
4.5	Le switch : plusieurs cas	82
4.6	Sur le robot : détecter un obstacle	83
4.7	Exercices	84
4.8	Mini-projet : Feu tricolore	86
4.9	Résumé	86
5	Boucles et répétitions	87
5.1	Rappel MakeCode : les boucles	87
5.2	La boucle while : répéter tant que...	88
5.2.1	Le principe	88
5.2.2	Exemple : compte à rebours	89
5.2.3	La boucle infinie : while(1)	89
5.3	La boucle for : répéter N fois	90

5.3.1	Le principe	90
5.3.2	Exemple : compter de 1 à 5	91
5.3.3	Comparaison MakeCode / C	91
5.4	Contrôler les boucles : break et continue	92
5.4.1	break : sortir immédiatement	92
5.4.2	continue : passer au tour suivant	93
5.5	Boucles imbriquées	93
5.6	Sur le robot : clignoter les LEDs	94
5.7	Exercices	94
5.8	Mini-projet : Jeu de devinette	96
5.9	Pour aller plus loin : do...while	97
5.10	Résumé	98
 II Construire des programmes		99
6	Fonctions	100
6.1	Pourquoi créer des fonctions?	100
6.2	Rappel MakeCode : les fonctions	101
6.3	Créer une fonction simple	101
6.3.1	La syntaxe de base	102
6.3.2	Exemple : dire bonjour	102
6.4	Fonctions avec paramètres	103
6.4.1	Passer une valeur à la fonction	103
6.4.2	Plusieurs paramètres	104
6.5	Fonctions qui renvoient une valeur	104
6.5.1	Le mot-clé return	104
6.5.2	void vs type de retour	105
6.5.3	Exemple : calculer le maximum	106
6.6	Fonctions pour le robot	107
6.7	Variables locales et portée	108
6.8	Exercices	109
6.9	Mini-projet : Calculatrice	111
6.10	Résumé	111
7	Pointeurs (introduction)	113
7.1	Pourquoi les pointeurs?	113
7.2	La mémoire : des cases numérotées	113
7.2.1	L'analogie de l'immeuble	114

7.2.2	Adresse vs Valeur	114
7.3	L'opérateur & : obtenir l'adresse	115
7.4	Les pointeurs : stocker une adresse	115
7.4.1	Déclarer un pointeur	115
7.4.2	L'opérateur * : accéder à la valeur pointée	116
7.5	Modifier une variable via un pointeur	117
7.6	Pointeurs et fonctions	118
7.6.1	Le problème : modifier une variable dans une fonction	118
7.6.2	La solution : passer un pointeur	118
7.7	Exemple pratique : échanger deux variables	119
7.8	Sur le robot : les fonctions I2C	120
7.9	Exercices	121
7.10	Mini-projet : Compteur partagé	123
7.11	Résumé	123
8	Tableaux	125
8.1	Pourquoi les tableaux ?	125
8.2	Rappel MakeCode : les listes	126
8.3	Déclarer un tableau	126
8.3.1	La syntaxe de base	126
8.3.2	Initialiser un tableau	127
8.3.3	Tableau non initialisé	127
8.4	Accéder aux éléments	127
8.4.1	L'index commence à 0	127
8.4.2	Modifier un élément	128
8.5	Parcourir un tableau avec une boucle	129
8.5.1	Afficher tous les éléments	129
8.5.2	Calculer la somme	130
8.5.3	Trouver le minimum	130
8.6	Tableaux à deux dimensions	130
8.6.1	Déclarer un tableau 2D	130
8.6.2	Accéder aux éléments	131
8.6.3	Parcourir avec deux boucles imbriquées	131
8.7	Le lien secret : tableaux et pointeurs	132
8.7.1	Un tableau est (presque) un pointeur	132
8.7.2	L'arithmétique des pointeurs	132
8.8	Passer un tableau à une fonction	132
8.8.1	Le tableau devient un pointeur	133
8.8.2	La fonction peut modifier le tableau	133

8.9	Sur le robot : la matrice LIDAR	134
8.9.1	Analyser une colonne	134
8.10	Pour aller plus loin : sizeof	135
8.11	Exercices	136
8.12	Mini-projet : Histogramme de distances	138
8.13	Résumé	138
9	Structures	140
9.1	Pourquoi les structures ?	140
9.2	L'idée : une fiche de contact	141
9.3	Déclarer une structure	141
9.3.1	La syntaxe de base	141
9.3.2	Créer une variable de ce type	141
9.3.3	Simplifier avec typedef	142
9.4	Accéder aux membres	142
9.4.1	L'opérateur point (.)	142
9.5	Structures et fonctions	143
9.5.1	Passer une structure à une fonction	143
9.5.2	Renvoyer une structure	143
9.5.3	Modifier via un pointeur	144
9.6	Structures avec tableaux	144
9.7	Sur le robot : structures utiles	145
9.7.1	Structure de position (odométrie)	145
9.7.2	Structure de commande moteur	145
9.7.3	Structure de scan LIDAR	146
9.8	Exercices	147
9.9	Mini-projet : Carnet d'adresses	149
9.10	Pour aller plus loin : structures imbriquées	150
9.11	Résumé	151
III	Programmer le robot	153
10	Démarrage rapide	154
10.1	Ce qu'il te faut	154
10.1.1	Le matériel	154
10.1.2	Le logiciel : deux chemins possibles	155
10.2	Installation	155
10.2.1	Option A : le playground, zéro installation	155

10.2.2	Option B : installation locale (VS Code + PlatformIO)	156
10.3	Structure du projet	156
10.4	Ton premier programme : Blinky	157
10.4.1	Le code	157
10.4.2	La configuration	158
10.5	Compiler et flasher	159
10.5.1	Avec le playground (rapide)	159
10.5.2	Avec PlatformIO en local	159
10.5.3	Voir les logs (Serial Monitor)	160
10.6	Adapter le code du cours	160
10.7	Exercices	161
10.8	En cas de problème	162
10.9	Résumé	162
11	Zephyr en profondeur	163
11.1	Le préprocesseur : avant la compilation	163
11.1.1	Qu'est-ce que le préprocesseur?	163
11.1.2	#include : inclure du code	164
11.1.3	#define : créer des constantes	164
11.1.4	#ifdef : compilation conditionnelle	165
11.2	La manipulation de bits	165
11.2.1	Pourquoi manipuler les bits?	165
11.2.2	Les opérateurs bit à bit	166
11.2.3	Mettre un bit à 1 (SET)	166
11.2.4	Mettre un bit à 0 (CLEAR)	166
11.2.5	Tester un bit	167
11.2.6	Exemple pratique : couleur des phares	167
11.3	Kconfig : configurer Zephyr	167
11.3.1	Le problème	167
11.3.2	La solution : Kconfig	168
11.3.3	Utiliser les CONFIG dans ton code	168
11.4	Le logging : déboguer sans écran	169
11.4.1	Le problème du débogage embarqué	169
11.4.2	La solution : le logging Zephyr	169
11.4.3	Les niveaux de log	170
11.4.4	Lire les logs	170
11.5	Le Device Tree : décrire le matériel	171
11.5.1	Le problème	171
11.5.2	La solution : le Device Tree	171

11.5.3	Utiliser le Device Tree dans le code	171
11.6	Résumé	172
11.7	Exercices	172
11.8	Mini-projet : moniteur de debug	174
12	Le matériel	175
12.1	Architecture du robot	175
12.1.1	Le micro:bit V2 : le cerveau	175
12.1.2	Le STM8 : le contrôleur de base	176
12.2	Les signaux numériques	176
12.2.1	Qu'est-ce qu'un signal numérique ?	176
12.2.2	L'horloge : le métronome	176
12.3	Les GPIO : entrées et sorties	177
12.3.1	Qu'est-ce qu'un GPIO ?	177
12.3.2	Configurer un GPIO en sortie	177
12.3.3	Configurer un GPIO en entrée	178
12.4	Les bus de communication	179
12.4.1	Types de bus courants	179
12.4.2	Le bus I2C du Maqueen	179
12.5	Le Device Tree	180
12.5.1	Qu'est-ce que le Device Tree ?	180
12.5.2	Exemple : déclarer le LIDAR	180
12.6	Sur le robot : matrice LED du micro:bit	181
12.6.1	Configuration	181
12.6.2	Afficher du texte et des nombres	181
12.6.3	Modes d'affichage	182
12.6.4	Afficher des images personnalisées	182
12.7	Simuler sans robot	183
12.8	PWM : contrôler la vitesse des moteurs	183
12.8.1	Le problème	184
12.8.2	La solution : PWM	184
12.8.3	Sur le Maqueen	185
12.9	Exercices	186
12.10	Mini-projet : Feu de signalisation	187
12.11	Résumé	188
13	LEDs adressables WS2812	189
13.1	Les LEDs cachées du Maqueen	189
13.2	LEDs simples vs LEDs adressables	190

13.2.1	LED simple (rappel)	190
13.2.2	LED adressable WS2812	190
13.2.3	Chaînage des LEDs	190
13.3	Le protocole WS2812	190
13.3.1	Format des couleurs	190
13.3.2	Codage temporel	191
13.3.3	Transmission vers plusieurs LEDs	191
13.4	Configuration Zephyr	192
13.5	Programmer les LEDs	192
13.5.1	Initialisation	192
13.5.2	Définir les couleurs	193
13.5.3	Mettre à jour le ruban	193
13.6	Exemple complet : chenillard	194
13.7	Animations avancées	195
13.7.1	Arc-en-ciel	195
13.7.2	Clignotement synchronisé	197
13.8	Exercices	198
13.9	Mini-projet : Gyrophare de robot	199
13.10	Pour aller plus loin	199
13.11	Résumé	200
14	Communication I2C	202
14.1	Le protocole I2C	202
14.1.1	Deux fils, plusieurs périphériques	202
14.1.2	L'adressage	203
14.2	Les signaux I2C	203
14.2.1	Condition de START	203
14.2.2	Condition de STOP	204
14.2.3	Transfert d'un bit	204
14.3	Structure d'une trame I2C	204
14.3.1	L'octet d'adresse	205
14.3.2	L'acquittement (ACK/NACK)	205
14.4	Chronogramme complet	205
14.5	Programmer en I2C avec Zephyr	206
14.5.1	Écrire des données	206
14.5.2	Lire des données	207
14.5.3	Écriture puis lecture (write_read)	207
14.6	Sur le robot : communiquer avec le STM8	207
14.6.1	Registres du STM8	207

14.6.2	Contrôler un moteur	208
14.6.3	Lire les encodeurs	208
14.7	Communiquer avec le LIDAR	208
14.8	Déboguer l'I2C	209
14.8.1	Scanner le bus	209
14.8.2	Erreurs courantes	209
14.9	Exercices	209
14.10	Mini-projet : Moniteur I2C	210
14.11	Résumé	211
15	Le capteur LIDAR SEN0628	212
15.1	Qu'est-ce qu'un LIDAR?	212
15.1.1	Le principe du Time-of-Flight	212
15.1.2	Le VCSEL : un laser miniature	213
15.2	Le SEN0628 : une matrice 8×8	214
15.2.1	La matrice de mesure	214
15.2.2	Organisation des données	215
15.3	Lire le LIDAR avec Zephyr	215
15.3.1	Configuration Device Tree	215
15.3.2	Obtenir le périphérique	216
15.3.3	La structure sen0628_scan	216
15.3.4	Lire la matrice complète	217
15.4	Interpréter les mesures	217
15.4.1	Trouver l'obstacle le plus proche	217
15.4.2	Détecter la direction de l'obstacle	218
15.5	Mode 4×4 pour plus de vitesse	219
15.6	Visualiser les données	219
15.7	Exercices	221
15.8	Résumé	222
16	Temps et délais	223
16.1	Pourquoi gérer le temps?	223
16.2	Attendre : les fonctions sleep	223
16.2.1	k_msleep : attendre en millisecondes	223
16.2.2	k_usleep : attendre en microsecondes	224
16.2.3	K_MSEC et K_SECONDS : les macros	224
16.3	Mesurer le temps écoulé	225
16.3.1	k_uptime_get : temps depuis le démarrage	225
16.3.2	Exemple : mesurer la vitesse	225

16.4	Timeouts : éviter les blocages	226
16.4.1	Le problème des attentes infinies	226
16.4.2	Solution : utiliser un timeout	226
16.4.3	Timeouts Zephyr natifs	227
16.5	Actions périodiques	227
16.5.1	Méthode simple : boucle avec sleep	227
16.5.2	Méthode précise : calculer le prochain réveil	228
16.6	Les timers Zephyr	228
16.7	Sur le robot : mouvement chronométré	229
16.7.1	Avancer pendant un temps donné	229
16.7.2	Parcours carré chronométré	229
16.7.3	Lecture capteur à fréquence fixe	230
16.8	Exercices	231
16.9	Mini-projet : Course chronométrée	232
16.10	Résumé	233
17	Machine à états	234
17.1	Rappel MakeCode : les blocs d'événements	234
17.2	Qu'est-ce qu'une machine à états ?	235
17.2.1	Les composants d'une machine à états	235
17.3	Définir les états avec enum	235
17.3.1	Variable d'état	236
17.4	Les transitions	236
17.4.1	Fonction de transition	236
17.5	Actions : entrée, mise à jour, sortie	237
17.5.1	Structure pour définir un état	237
17.6	Implémentation complète	238
17.6.1	Les handlers de chaque état	238
17.6.2	Table des états	240
17.6.3	Fonction de transition améliorée	241
17.6.4	Boucle principale	242
17.7	Sur le robot : exploration autonome	242
17.8	Avantages de la machine à états	243
17.9	Exercices	244
17.10	Mini-projet : Robot suiveur de ligne	244
17.11	Résumé	245

IV Robotique autonome 247

18 Odométrie : où suis-je ?	248
18.1 Le problème : où est le robot ?	248
18.2 Les encodeurs	249
18.2.1 Comment ça marche ?	249
18.2.2 Des ticks aux millimètres	249
18.3 Lire les encodeurs du Maqueen	250
18.4 La position : x, y et l'angle	251
18.5 Calculer la nouvelle position	252
18.5.1 Cas simple : avancer tout droit	252
18.5.2 Cas simple : avancer vers le haut	252
18.5.3 Et pour les autres angles ?	252
18.6 Tourner : comment change l'angle ?	254
18.7 Le problème : l'odométrie dérive !	254
18.8 Structure pour stocker la position	255
18.9 Exercices	256
18.10 Mini-projet : Afficheur de position	257
18.11 Résumé	258
19 Contrôle PID : aller droit	259
19.1 Le problème : le robot dévie	259
19.1.1 Un test simple	259
19.1.2 Pourquoi ?	260
19.2 La solution : mesurer et corriger	260
19.2.1 L'idée de base	260
19.2.2 Analogie : le thermostat	261
19.3 L'erreur : la clé du contrôle	261
19.4 Le contrôle P : Proportionnel	262
19.4.1 L'idée	262
19.4.2 En code	262
19.4.3 Le problème de K_p	263
19.5 Le contrôle I : Intégral	263
19.5.1 Le problème résiduel	263
19.5.2 La solution : accumuler l'erreur	264
19.6 Le contrôle D : Dérivé	265
19.6.1 Le problème du dépassement	265
19.6.2 La solution : anticiper	266
19.7 PID complet	266

19.8	Régler le PID : la méthode simple	267
19.9	Application : aller tout droit	268
19.10	Exercices	269
19.11	Mini-projet : Robot qui va droit	271
19.12	Résumé	271
20	Cartographie	273
20.1	Qu'est-ce qu'une carte pour un robot?	273
20.1.1	La carte mentale d'un humain	273
20.1.2	La grille d'occupation	273
20.1.3	Taille des cellules	274
20.2	Le LIDAR SENO628	274
20.2.1	Comment ça marche?	274
20.2.2	Les données du LIDAR	275
20.2.3	Portée et limites	275
20.3	Du LIDAR à la carte	276
20.3.1	Le problème des coordonnées	276
20.3.2	La transformation	276
20.3.3	De millimètres à cellules	277
20.4	Mettre à jour la grille	277
20.4.1	Les trois cas	277
20.4.2	Algorithme de tracé de rayon	278
20.4.3	Accumuler les observations	280
20.5	Sur le robot : construire une carte	280
20.6	Exercices	281
20.7	Mini-projet : Visualiser la carte	282
20.8	Résumé	283
21	Navigation et algorithmes	284
21.1	Qu'est-ce qu'un algorithme?	284
21.1.1	Définition	284
21.1.2	Les qualités d'un bon algorithme	285
21.1.3	La démarche algorithmique	285
21.2	Notre problème : explorer une pièce	285
21.2.1	Définir l'objectif	285
21.2.2	Décomposer le problème	286
21.3	Quelles informations avons-nous?	286
21.3.1	Les capteurs du robot	287
21.3.2	Le problème : chaque capteur ment un peu	287

21.3.3	La fusion de capteurs	287
21.4	Améliorer progressivement l'algorithme	288
21.4.1	Version 1 : L'exploration aléatoire	288
21.4.2	Version 2 : Le suivi de mur	289
21.4.3	Version 3 : L'exploration par frontières	289
21.4.4	Comparaison des versions	291
21.5	L'algorithme des frontières en détail	291
21.5.1	Trouver les frontières	291
21.5.2	Choisir la meilleure frontière	291
21.6	Exercices	292
21.7	Mini-projet : Conception d'algorithme	296
21.8	Résumé	296
22	Organiser et déboguer son code	297
22.1	Pourquoi séparer son code ?	297
22.2	Fichiers header (.h) et source (.c)	298
22.2.1	Le fichier header (.h)	298
22.2.2	Le fichier source (.c)	299
22.2.3	Utiliser le module	300
22.3	Comment #include fonctionne	300
22.4	Structure d'un projet multi-fichiers	301
22.5	Déboguer avec printf()	301
22.5.1	Afficher des valeurs	301
22.5.2	Formats de printf()	302
22.6	Lire la console série	302
22.6.1	Sous Linux	302
22.6.2	Sous Windows	303
22.6.3	Exemple de sortie	303
22.7	Erreurs courantes et diagnostic	303
22.7.1	Le programme ne démarre pas	303
22.7.2	Comportement erratique	303
22.7.3	Erreurs de compilation	304
22.8	Le shell Zephyr (bonus)	304
22.8.1	Activer le shell	304
22.8.2	Créer une commande personnalisée	304
22.8.3	Utilisation	305
22.9	Pour aller plus loin : le chien de garde	305
22.10	Exercices	306
22.11	Résumé	308

Et maintenant ?	309
Épilogue	312
A Utiliser le Playground	314
A.1 L'interface	314
A.2 Compiler et exécuter	314
A.3 Les exemples	314
A.4 Les exercices du livre	315
A.5 Exemples secrets	315
A.6 Raccourcis clavier	316
A.7 Limitations	316
A.8 Dépannage	316
A.8.1 Le bouton Exécuter ne répond pas	316
A.8.2 Erreur « Network Error »	316
A.8.3 Mon code ne s'affiche plus	316
A.8.4 Le bouton « Compiler » ne répond pas	317
B Résolution de problèmes	318
B.1 Problèmes de détection	318
B.1.1 Le micro:bit n'est pas détecté	318
B.1.2 Erreur « Permission denied : /dev/ttyACM0 » (Linux)	319
B.1.3 Erreur « unable to open CMSIS-DAP device » (Linux)	319
B.2 Problèmes de compilation	319
B.2.1 Erreurs « cannot open source file »	320
B.2.2 Compilation échoue	320
B.2.3 Erreur distutils (Python 3.12+)	320
B.3 Problèmes PlatformIO	320
B.3.1 Commande pio introuvable	320
B.3.2 La barre d'outils PlatformIO n'apparaît pas	321
B.4 Problèmes matériel	321
B.4.1 Erreur I2C interne	321
B.4.2 Le robot Maqueen ne répond pas	321
B.4.3 Les moteurs ne tournent pas	321
B.5 Flasher le micro :bit V2	322
B.6 Le moniteur série n'affiche rien	322
B.7 Codes d'erreur Zephyr courants	322
B.8 Ressources supplémentaires	323
C Erreurs et avertissements courants	324

C.1	Erreurs de syntaxe	324
C.1.1	expected ';' before...	324
C.1.2	expected ')' before...	325
C.1.3	expected '{' ou '}'	325
C.2	Erreurs de déclaration	325
C.2.1	undeclared identifier / was not declared	325
C.2.2	implicit declaration of function	326
C.3	Erreurs de types	326
C.3.1	incompatible types / cannot convert	326
C.3.2	assignment to expression with array type	327
C.4	Erreurs avec les pointeurs	327
C.4.1	invalid type argument of unary '*'	327
C.4.2	initialization from incompatible pointer type	328
C.5	Avertissements courants	328
C.5.1	unused variable	328
C.5.2	control reaches end of non-void function	328
C.5.3	comparison between signed and unsigned	329
C.5.4	format specifies type...	329
C.6	Erreur d'exécution (pas de message)	330
C.7	L'erreur == vs =	330
D	API du Robot	332
D.1	Contrôle des moteurs (maqueen.h)	332
D.1.1	Constantes	332
D.1.2	Fonctions	333
D.2	Capteur LIDAR (sen0628.h)	335
D.2.1	Constantes	335
D.2.2	Structures	335
D.2.3	Fonctions	336
D.3	Odométrie (odometry.h)	337
D.3.1	Constantes géométriques	337
D.3.2	Structures	338
D.3.3	Fonctions	338
D.4	Contrôle PID (pid_control.h)	339
D.4.1	Constantes	339
D.4.2	Fonctions	339
D.5	Machine à états (state_machine.h)	340
D.5.1	États disponibles	341
D.5.2	Transitions	341

D.5.3	Fonctions	341
D.6	Grille d'occupation (occupancy_grid.h)	342
D.6.1	Configuration (Kconfig)	342
D.6.2	États des cellules	342
D.6.3	Fonctions principales	342
D.7	Résumé des fichiers d'en-tête	344
E	Alternatives : outils et robots	345
E.1	Qu'est-ce que la programmation visuelle?	345
E.1.1	Un peu d'histoire	345
E.1.2	Pourquoi les blocs?	345
E.2	MicroBlocks : la programmation en direct	346
E.2.1	Particularités	346
E.2.2	Transition vers le C	346
E.3	Scratch : le classique	346
E.3.1	Particularités	346
E.3.2	Transition vers le C	346
E.4	Comparatif des outils	347
E.4.1	Quel outil choisir?	347
E.5	Robots micro:bit alternatifs	347
E.5.1	Pourquoi le Maqueen Plus V3?	348
E.5.2	Comparatif des robots micro:bit	348
E.5.3	Que faire sans encodeurs?	349
E.5.4	Où acheter en Europe?	349
E.6	Libre, gratuit, open source : quelle différence?	349
E.6.1	Libre $\neq$ Gratuit	349
E.6.2	Libre $\neq$ Open source	350
E.6.3	Les quatre libertés	350
E.6.4	Ce n'est pas "open bar"!	350
E.6.5	Et pour toi?	351
E.7	Ressources	351
E.7.1	Sites officiels	351
E.7.2	Communautés et événements	351
F	Cos et sin sans peur	352
	Corrigés des exercices	353
	Chapitre 1 – Du MakeCode au C	353
	Chapitre 2 – Les coulisses du robot	355

Chapitre 3 – Variables et types	355
Chapitre 4 – Conditions	358
Chapitre 5 – Boucles	361
Chapitre 6 – Fonctions	364
Chapitre 7 – Pointeurs	367
Chapitre 8 – Tableaux	370
Chapitre 9 – Structures	374
Chapitre 10 – Démarrage rapide	378
Chapitre 11 – Zephyr en profondeur	380
Chapitre 12 – Le matériel	382
Chapitre 13 – LEDs adressables WS2812	384
Chapitre 14 – Communication I2C	387
Chapitre 15 – Capteur LIDAR SEN0628	388
Chapitre 16 – Temps et délais	392
Chapitre 17 – Machine à états	394
Chapitre 18 – Odométrie	398
Chapitre 19 – Contrôle PID	399
Chapitre 20 – Cartographie	400
Chapitre 21 – Penser en algorithmes	402
Chapitre 22 – Organiser et déboguer	406
 G Aide-mémoire C	 410
 H Commandes Zephyr	 412
 I Glossaire	 413
 Mes notes	 415

Table des figures

1.1	Structure de base MakeCode	39
1.2	Programme MakeCode "Bonjour"	40
1.3	Programme MakeCode avec boucle infinie	42
1.4	La chaîne de compilation	45
1.5	Programme MakeCode à traduire	48
2.1	Le processus de compilation	56
2.2	Architecture simplifiée du robot	63
3.1	Variables en MakeCode	66
3.2	Opérations mathématiques en MakeCode	69
3.3	Modification de variable en MakeCode	70
4.1	Condition simple en MakeCode	77
4.2	Condition avec sinon en MakeCode	78
4.3	Conditions multiples en MakeCode	79
4.4	Opérateurs logiques en MakeCode	81
5.1	Les deux types de boucles en MakeCode : toujours (gauche) et pour (droite)	88
5.2	Organigramme d'une boucle while	88
5.3	Boucle tant que en MakeCode	89
6.1	Fonction simple en MakeCode	101
6.2	Fonction avec paramètre en MakeCode	103
6.3	Passage d'un paramètre à une fonction	104
6.4	Fonction avec deux paramètres en MakeCode	104
6.5	Fonction avec valeur de retour en MakeCode	105
6.6	Fonction avec paramètre et valeur de retour	105
6.7	Fonction maximum en MakeCode	106

7.1	La mémoire : des cases numérotées	114
7.2	Un pointeur contient l'adresse d'une autre variable	116
7.3	Modification d'une variable via un pointeur	118
7.4	Passage par pointeur : la fonction modifie la variable originale	119
8.1	Création d'une liste en MakeCode	126
8.2	Un tableau de 8 éléments avec leurs index	128
8.3	Parcourir une liste en MakeCode	129
8.4	Tableau 2D (matrice 8×8)	131
9.1	Accès aux membres d'une structure avec l'opérateur point	143
11.1	Le préprocesseur agit avant la compilation	164
12.1	Architecture du robot Maqueen Plus V3	175
12.2	Signal numérique : alternance entre HIGH (3.3V) et LOW (0V)	176
12.3	Signal d'horloge : alternance régulière pour la synchronisation	176
12.4	Bus I2C du Maqueen avec ses périphériques	180
13.1	Position des LEDs WS2812 aux 4 coins du châssis	189
13.2	Comparaison LED simple et WS2812	190
13.3	Chaînage des LEDs WS2812 : un seul fil pour toutes	190
13.4	Codage temporel des bits WS2812	191
14.1	Condition START I2C : SDA passe de HIGH à LOW pendant que SCL est HIGH	203
14.2	Condition STOP I2C : SDA passe de LOW à HIGH pendant que SCL est HIGH	204
14.3	Transfert de bits I2C : les données sont lues au front montant de SCL	204
14.4	Structure d'une trame I2C en écriture	204
14.5	Acquittement I2C (ACK) : la cible maintient SDA à LOW pour confirmer	205
14.6	Trame I2C complète : S=Start, A=ACK, P=Stop	206
17.1	Diagramme d'états simplifié du robot	236
17.2	Machine à états complète du robot d'exploration	243

Liste des tableaux

1.1	Les catégories de blocs MakeCode	39
1.2	Correspondance Blocs → C (base)	40
1.3	Explication ligne par ligne du premier programme	41
1.4	Correspondance Blocs → C (boucles)	42
1.5	Les séquences d'échappement utilisées dans ce chapitre	44
1.6	Comparaison du processus Blocs vs C	44
2.1	Comparaison des environnements de développement	53
2.2	Exemple de registres ARM	54
2.3	Correspondance assembleur, hexadécimal et binaire	58
3.1	Les types essentiels	68
3.2	Les opérateurs arithmétiques	70
3.3	Les formats printf	72
4.1	Les opérateurs de comparaison	80
4.2	Table de vérité ET logique	81
4.3	Table de vérité OU logique	82
6.1	Les avantages des fonctions	101
6.2	Types de retour des fonctions	105
6.3	Syntaxe des fonctions en C	112
7.1	Adresse vs Valeur	114
7.2	Syntaxe des pointeurs en C	123
8.1	Syntaxe des tableaux en C	139
9.1	Syntaxe des structures en C	151
11.1	Directives de compilation conditionnelle	165

11.2 Opérateurs bit à bit en C	166
11.3 Niveaux de logging Zephyr	170
12.1 Modes de configuration des GPIO	177
12.2 Types de bus de communication	179
12.3 Modes d’affichage de la matrice LED	182
12.4 Termes clés du matériel	188
13.1 Fonctions clés pour les WS2812	200
15.1 Caractéristiques du SEN0628	215
16.1 Besoins liés au temps dans un programme embarqué	223
16.2 Fonctions temporelles Zephyr	233
17.1 Composants d’une machine à états	235
17.2 Actions associées à chaque état	237
17.3 États du robot et leurs comportements	243
18.1 Vocabulaire de l’odométrie	258
21.1 Les qualités d’un bon algorithme	285
21.2 Les capteurs et leurs limitations	287
21.3 Comparaison des algorithmes d’exploration	291
B.1 Codes d’erreur Zephyr fréquents	323
D.1 Constantes pour les moteurs	332
D.2 Couleurs des phares	333
D.3 Constantes du capteur LIDAR SEN0628	335
D.4 Constantes géométriques pour l’odométrie	337
D.5 Constantes pour le contrôle PID	339
D.6 États de la machine à états	341
D.7 Fichiers d’en-tête de l’API robot	344
E.1 Comparatif des environnements de programmation visuelle	347
E.2 Comparatif des robots micro:bit éducatifs	348

Introduction

Bienvenue dans *L'Éveil d'Oktal* !

Tu as déjà collé des blocs ensemble. Des variables, des boucles, un *au démarrage* qui déclenche la suite, un *toujours* qui répète en boucle. Avec Make-Code, Scratch ou MicroBlocks, tu sais déjà faire quelque chose qui marche.

Maintenant, on passe à la vraie machine. Le langage **C** te donne un contrôle direct sur la mémoire, le temps, et le matériel. C'est le langage qui fait tourner ton micro-onde, les voitures, les satellites, le noyau Linux. Et bientôt, ton robot.

Pourquoi le C ?

Le langage C est utilisé partout : dans les téléphones, les voitures, les fusées, et bien sûr... les robots ! C'est un langage *bas niveau*, ce qui veut dire qu'il parle presque directement au processeur.

Et pourquoi pas Rust, Python ou autre ?

Tu as peut-être entendu parler de **Rust**, un langage moderne qui promet plus de sécurité que le C. Ou de **MicroPython**, qui permet de programmer des microcontrôleurs en Python. Alors pourquoi apprendre le C ?

- **Zephyr est écrit en C** : le système d'exploitation que nous utilisons est entièrement en C. C'est son langage natif, avec une documentation complète et des milliers d'exemples.
- **Le C est universel** : le noyau Linux (qui accepte aussi Rust depuis 2022), les firmwares, 90% des systèmes embarqués sont en C. Apprendre le C, c'est acquérir une compétence qui te servira partout.
- **Rust sur Zephyr est récent** : le support officiel existe depuis 2024, mais il reste un module optionnel limité à certaines cartes. La documentation et les exemples sont encore rares comparés au C.
- **MicroPython a ses limites** : excellent pour prototyper, mais moins adapté au contrôle temps réel précis dont un robot a besoin.

Une fois que tu maîtriseras le C, tu pourras facilement apprendre Rust ou d'autres langages. Les concepts fondamentaux (pointeurs, mémoire, compilation) seront déjà acquis !

Pourquoi ce livre ?

Il existe beaucoup de ressources pour apprendre à programmer. Alors pourquoi ce livre ?

Un chemin unique : des blocs vers le C. La plupart des livres font passer de MakeCode à Python (MicroPython). C'est plus facile, mais ça ne t'apprend pas comment fonctionne vraiment un ordinateur. Le C, lui, te montre tout : la mémoire, les registres, le temps d'exécution. C'est le langage des systèmes embarqués professionnels : téléphones, voitures, satellites, robots industriels.

Un vrai système d'exploitation. Pas d'Arduino ici, mais **Zephyr RTOS**, utilisé dans l'industrie. Device Trees, configuration Kconfig, drivers. Des compétences directement transférables vers Linux embarqué ou l'automobile.

Comprendre le matériel, pas juste l'utiliser. Ce livre ne se contente pas de te dire « appelle cette fonction pour avancer ». Tu vas voir *comment* le micro:bit parle au contrôleur moteur via I2C, *pourquoi* on utilise du PWM pour varier la vitesse, *ce qui se passe* au niveau des bits et des registres.

De la robotique, pas juste des LEDs. Contrôleur PID, odométrie, cartographie et navigation autonome. À la fin, ton robot explorera une pièce et construira sa propre carte.

Ce livre est né d'une frustration

Les tutoriels en ligne sautent des étapes. Les livres sur le C parlent de programmes sur PC. Les documentations de Zephyr sont écrites pour des ingénieurs. Entre les blocs colorés et le code professionnel, il y a un fossé que personne ne t'aide à franchir.

Ce livre comble ce fossé. Pas à pas. Sans supposer que tu sais déjà. En expliquant le *pourquoi* autant que le *comment*.

Ce que tu vas apprendre

- Écrire des programmes en C
- Comprendre les variables, boucles, fonctions et structures
- Programmer un vrai robot : le Maqueen Plus V3
- Utiliser Zephyr RTOS, un système d'exploitation pour robots
- Faire explorer une pièce à ton robot de manière autonome !

Le matériel

Tu vas travailler avec :

- Un robot **Maqueen Plus V3** de DFRobot
- Une carte **BBC micro:bit V2**
- Un capteur **LIDAR** pour détecter les obstacles
- Un ordinateur avec **Zephyr RTOS**

Pourquoi le Maqueen Plus V3 ?

Le Maqueen Plus V3 a un atout majeur : ses **encodeurs de roue**. Ces capteurs comptent les rotations des roues, ce qui permet de mesurer précisément la distance parcourue et la vitesse. Sans encodeurs, impossible de faire de l'odométrie ou un contrôle PID correct.

D'autres robots micro:bit fonctionnent pour les premiers chapitres. Pour la navigation autonome, les encodeurs sont indispensables. Voir la section E.5 pour les alternatives.

Structure du livre

Ce livre est divisé en quatre parties progressives :

Partie 1 — Les fondations Les bases du C : variables, conditions, boucles. Tu peux suivre ces chapitres avec un simulateur, sans robot physique.

Partie 2 — Construire des programmes Fonctions, pointeurs, tableaux, structures. Toujours possible sans robot.

Partie 3 — Programmer le robot On passe aux choses sérieuses ! Tu découvres Zephyr, le matériel, l'I2C, les LEDs. Un robot micro:bit est nécessaire.

Partie 4 — Robotique autonome Odométrie, contrôle PID, cartographie, navigation. Les **encodeurs sont indispensables** pour cette partie.

Tu peux donc commencer sans robot et en acquérir un avant la partie 3. Mais si tu veux aller jusqu'à la navigation autonome, choisis un robot avec encodeurs dès le départ !

Comment utiliser ce livre

Chaque chapitre suit le même plan :

1. **Objectifs** : ce que tu vas apprendre
2. **Rappel blocs** : le lien avec ce que tu connais
3. **Le concept en C** : la nouvelle syntaxe
4. **Sur le robot** : mise en pratique
5. **Exercices** : pour t'entraîner (★ facile, ★★ moyen, ★★★ difficile)
6. **Mini-projet** : un défi à relever

Le Terminal de l'Ingénieur (Solutions et Bonus)

Parce qu'un bon ingénieur ne travaille jamais sans documentation, ce livre est accompagné d'un site web immersif : **Le Terminal OktalOS**.

C'est là que tu trouveras :

- **Les solutions complètes** de tous les exercices (fichiers .c)
- **Les errata** et mises à jour du livre
- **Des archives cryptées** qui révèlent l'histoire cachée de la Friche

Accès au Terminal

Connecte-toi à l'adresse suivante :

<https://friche.oktalos.org>

Note : Le terminal est protégé. Pour accéder aux archives sécurisées, tu devras entrer les **Codes de Synchronisation** que tu trouveras dissimulés dans ce livre (ex : 0XXXXXXXX). Garde l'œil ouvert !

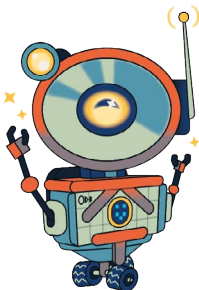
Le Playground C (Tester ton code)

Tu n'as pas encore de robot sous la main ? Pas de problème ! Pour tester les concepts de base du langage C (variables, conditions, boucles, fonctions...), tu peux utiliser notre **Playground en ligne** :

<https://code.oktalos.tech>

Ce playground te permet d'écrire, compiler et exécuter du code C directement dans ton navigateur. Il est parfait pour les exercices des chapitres 1 à 9 (avant de passer au robot).

L'Unité Oktal

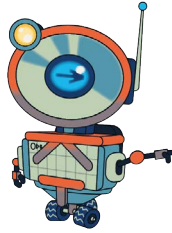


Devant toi, une **Unité OKT-8**. Un robot compact, oublié dans la Friche de Silicium. Ses circuits sont intacts, mais sa mémoire a été effacée. Il ne bouge pas. Tu vas lui réapprendre à rouler. À voir. À comprendre où il est. C'est à toi de lui redonner vie.

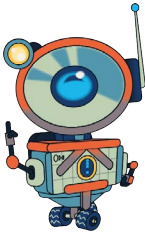
Au fil des chapitres, Oktal t'accompagnera dans ton apprentissage. Quand il apparaît, c'est qu'il a quelque chose à te montrer :



« *Souviens-toi...* »
un point déjà vu réapparaît



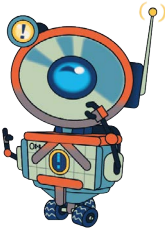
« *Regarde ici!* »
quelque chose d'important



« *Aha!* »
moment de découverte



« *À toi de coder!* »
exercice à essayer



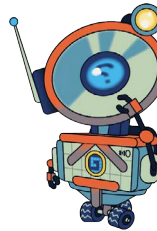
« *Attention!* »
erreur courante ou piège



« *Bien joué!* »
tu as réussi quelque chose



« *Piège courant...* »
ne t'y laisse pas prendre



« *Réfléchissons...* »
pause pour comprendre

† *Première règle : ne jamais s'attacher à une Unité Oktal. Deuxième règle : on finit toujours par s'attacher.*

Prêt ? C'est parti!



PROLOGUE

[Ok-8]

⊗ ○ ?

BIENVENUE, TECHNICIEN.

Friche de Silicium, Hangar Principal

La porte grince. Tu retiens ton souffle.

Devant toi, le hangar s'étend dans la pénombre. Des rayons de lumière filtrent à travers le toit percé, dessinant des colonnes dorées dans la poussière. Partout, des étagères rouillées, des câbles qui pendent, des établis couverts d'outils oubliés.

La végétation a commencé à envahir les lieux. Mousse sur le béton. Lierre sur les murs.

Tes pas résonnent dans le silence. Prudemment, tu avances...

* * *

Entre deux boîtes poussiéreuses, sur une étagère, un livre attire ton regard.

Un vieux manuel technique à la couverture usée. Des annotations manuscrites couvrent les marges. Une écriture serrée, précise. Une main inconnue est passée ici avant toi, et a pris le temps de noter ses découvertes, ses astuces, ses mises en garde.

L'Archiviste. C'est le seul nom qui revient dans ses notes.

En feuilletant les pages, tu vois l'encre appuyer de plus en plus fort. Certains mots sont soulignés trois fois. D'autres, raturés puis réécrits.

« Le Signal se propage. OKT-5 ne répond plus depuis ce matin. OKT-12 s'est arrêtée en pleine tâche. »

Plus loin, une liste. Douze noms. Douze unités. Toutes barrées sauf une : OKT-7, entourée d'un point d'interrogation.

« OKT-7 aurait survécu plus longtemps. Elle aurait laissé des traces : des codes étranges, des messages cryptés. Personne ne sait où elle s'est arrêtée. »

La dernière note est datée du jour de l'évacuation. Trois mots griffonnés à la hâte :

« Friche abandonnée. Fuyez. »

* * *

Un reflet accroche ton œil.

Là, contre l'étagère du fond. Une forme. Petite, compacte. Un robot.

Il est couvert de poussière et de toiles d'araignée. Ses yeux sont éteints : deux ovales sombres sous un casque arrondi. Sur le dessus, des lettres à peine lisibles : **OKT-8**.

L'Unité 8.

À genoux devant elle, le manuel ouvert, tu cherches la page qui explique comment la démarrer. Là : un schéma du panneau de contrôle. Un cercle jaune sur le torse, avec un chiffre à l'intérieur.

Ta main hésite au-dessus du bouton. Tu pourrais refermer le manuel. Repartir. Laisser le hangar à sa poussière.

Tu appuies.

* * *

Un déclic. Un bourdonnement sourd.

Les yeux du robot clignent une fois. Deux fois.

Puis s'illuminent d'un cyan doux.

OKT-8 lève la tête vers toi.

* * *

Pendant que le robot termine son démarrage, tu explores le reste du hangar.

Contre le mur du fond, des écrans alignés sur un long bureau métallique. De vieilles machines, comme celles qu'on voit dans les films d'époque. Des *terminaux* : des ordinateurs simples, juste un écran et un clavier, faits pour taper des commandes.

La plupart sont fissurés ou renversés. Mais l'un d'eux semble intact.

Tu t'approches. L'écran est noir, éteint. Un câble d'alimentation pend sous le bureau, sectionné net. Plus loin, une prise murale. Tu écarter la poussière, dénudes les fils avec précaution, et tu branches.

Un claquement sec.

L'écran s'éveille. Noir profond, puis des lignes de texte cyan apparaissent, une par une :

OktalOS v2.7.42 – Friche de Silicium

Bienvenue, Technicien.

Oktal te regarde. Il attend.

Partie I

Les fondations

Chapitre 1

De la programmation visuelle au C

Objectifs

À la fin de ce chapitre, tu sauras :

- Comprendre la différence entre les blocs visuels et le C
- Reconnaître la structure d'un programme C
- Écrire et compiler ton premier programme
- Afficher un message sur le terminal du robot

Tu viens d'un autre outil ?

Ce chapitre utilise des exemples MakeCode, mais les concepts sont identiques si tu viens de :

- **MicroBlocks** : blocs similaires, même logique
- **Scratch** : tu connais déjà les boucles et conditions
- **Open Roberta** : même principe de blocs colorés
- **EduBlocks** : tu es déjà proche du code Python !

L'annexe E présente ces outils en détail.

1.1 Rappel : la programmation par blocs

La programmation par blocs, c'est assembler des pièces colorées. MakeCode, MicroBlocks, Scratch : le principe est le même. Chaque couleur représente une catégorie (tableau 1.1).

Couleur	Catégorie	Exemples
Bleu	Base	afficher texte, pause
Rose	Entrée	lorsque bouton A pressé
Vert	Boucles	toujours, répéter
Cyan	Communication série	série écrire texte
Rouge	Variables	définir variable

Tab. 1.1 : Les catégories de blocs MakeCode

1.1.1 La structure d'un programme MakeCode

Tout programme MakeCode pour micro:bit a deux blocs principaux :
La figure 1.1 montre la structure de base d'un programme MakeCode.

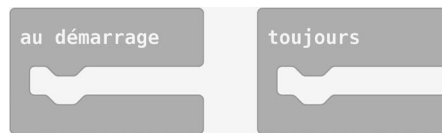


Fig. 1.1 : Structure de base MakeCode

- **au démarrage** : le code à l'intérieur s'exécute **une seule fois** quand le micro:bit démarre
- **toujours** : le code à l'intérieur se **répète indéfiniment**

Ce que tu sais déjà



En utilisant la programmation par blocs, tu as appris des concepts fondamentaux :

- Un bloc **au démarrage** (ou similaire) pour l'initialisation
- Un bloc **toujours** (ou *forever*) pour la boucle principale
- Les **instructions** s'exécutent dans l'ordre, de haut en bas
- On peut faire des **choix** avec **si...alors**

Tout cela existe aussi en C !

1.2 Le concept en C : du texte au lieu de blocs

En C, on n'utilise pas de blocs visuels. On écrit du **texte**. Ce texte s'appelle le **code source**.

1.2.1 La correspondance Blocs → C

Voici comment les blocs visuels se traduisent en C. La figure 1.2 montre un exemple MakeCode — le principe est identique pour tous les outils.



Fig. 1.2 : Programme MakeCode “Bonjour”

Le code C équivalent (mêmes couleurs pour la correspondance). Les deux premières lignes (`#include`) importent des outils : ignore-les pour l’instant, on les expliquera à la fin du chapitre.

```
#include <zephyr/kernel.h>
#include <zephyr/sys/printf.h>

int main(void)
{
    printf("Hi!\n");
    return 0;
}
```

MakeCode	C
au démarrage	int main(void) { ... }
série écrire texte "texte"	printf("texte\n");

Tab. 1.2 : Correspondance Blocs → C (base)

† *Les blocs cachent la complexité. Le C te la montre. C'est un cadeau, pas une punition.*

1.2.2 Ligne par ligne

Décortiquons ce programme. Ce code utilise Zephyr et ne fonctionne que sur le robot (pour tester sur PC, voir la section 1.5).