

Table des matières

À propos des auteurs	xii
Préface	xiii
Introduction	xv
1. Organisation de cet ouvrage	xv
2. À qui s'adresse cet ouvrage ?.....	xvi
3. Code source et licence	xvi
4. Remerciements	xvii

Quelques concepts de base 1

1. Établir sa feuille de route	2
1. Trop d'analyse tue l'analyse	3
2. Faites le test	4
2. Écrire un code compréhensible	5
3. Choisissez votre langue	6
4. Une déclaration par ligne	7
5. Choisissez des noms explicites	8
6. Une ligne = une instruction	10
7. Respectez des règles d'indentation strictes	11
8. Respectez les conventions	12
9. Les mêmes règles pour tout le projet	13
10. La solution la plus simple est toujours la moins compliquée	14
11. DRY : Don't Repeat Yourself.....	14
12. Préférez les énumérations	15
Marquez la fin des valeurs énumérées	17
Le fin du fin : les énumérations fortement typées.....	18
13. Préférez les tableaux à une dimension	22
En résumé	25
3. Du bon usage des commentaires	26
14. Un commentaire ne doit pas paraphraser le code	26
15. Un commentaire ne sert pas à marquer les étapes	27
16. Les commentaires à visée pédagogique	27
17. Les commentaires sous forme de <i>cartouche</i>	28
Pourquoi cette exception ?.....	28
Écrire un bon cartouche	29

Un petit exemple vaut mieux qu'un long discours	31
18. Quelques exceptions	32
4. Éviter les effets de bord	34
5. La pile et le tas	37
19. La pile, pour les données <i>automatiques</i>	37
20. Dans quel ordre ?.....	37
21. Mais quelle est cette portée ?.....	38
22. Pour les données <i>dynamiques</i>	39
6. Mon pote, le compilateur	40
7. NIH, ou l'art de réinventer la roue	41
Les fondements de la programmation orientée objet	42
8. Les grands principes de la POO	43
1. La loi de Déméter	44
Un exemple pour bien comprendre	44
Pourquoi se passer des accesseurs ?.....	45
Les mutateurs, c'est encore pire	47
2. Cinq piliers, c'est du SOLID	49
3. SRP : Le principe de la responsabilité unique	49
Méfiez-vous des termes trop vagues	50
4. OCP : Le principe ouvert/fermé	52
Fermé aux modifications ?.....	52
Mais ouvert aux évolutions ?.....	52
Comment y arriver ?.....	53
Le RTTI ne sert pas aux tests de types	53
Quelle solution alors ?.....	55
Mais comme rien n'est simple en ce bas monde.....	55
5. LSP : Le principe de substitution de Liskov	56
Comprendre LSP	57
L'héritage public : une relation EST-UN.....	59
Un principe de conception	59
Se poser les bonnes questions au bon moment	60
LSP et la programmation par contrat : le dernier bastion.....	61
Fonction publique == Propriété == Invariant	62
Changer la visibilité des fonctions membres	62
6. ISP : Le principe de ségrégation des interfaces	66

Une interface, qu'est-ce que c'est ?.....	66
Que nous dit l'ISP ?.....	67
D'accord, mais on fait comment ?.....	70
ISP pour éviter l'héritage en losange	71
Une classe template peut-être ?.....	73
7. DIP : Le principe d'inversion des dépendances	75
8. Point trop n'en faut	78
9. Les bases de la programmation par contrat.....	81
9. Préconditions et postconditions	81
10. Invariant ? kesako ?.....	82
11. Un développeur est coupable, mais lequel ?.....	84
12. Invariants <i>structurels</i> et invariants <i>contextuels</i>	85
13. La granularité des invariants	86
14. Tirer parti de la programmation par contrat	88
À la compilation, si faire se peut	88
Le constructeur, une aubaine	90
Au <i>runtime</i> , si pas le choix	92
15. Assertion ou exception ?.....	95
Les assertions	99
Les exceptions	100
10. Sur les pas de l'Xtrem Programming.....	102
16. YAGNI et KISS... mais pas trop	102
YAGNI : You Ain't Gonna Need It.....	102
KISS : Keep It Simple, Stupid.....	103
17. De l'importance des tests unitaires	104
Un projet <i>parallèle</i>	105
Commencez-les le plus tôt possible	106
Les fonctionnalités de base avant tout	107
Testez surtout les cas <i>border line</i>	108
Testez tout ce qui est testable	108
Tous les tests unitaires doivent passer	109
Il ne s'agit que d'une bonne indication	110
Parce qu'on ne peut pas penser à tout	112
Ni suppression ni modification	112
Et le C++ dans tout ça ?.....	114
11. Renoncer aux habitudes du C	115

1. Un tableau d'éléments ? C'est <code>std::vector</code>	116
2. Une chaîne de caractères ? C'est <code>std::string</code>	116
3. Le flux d'entrée par défaut ? C'est <code>std::cin</code>	117
4. Utiliser un fichier ? C'est <code>std::fstream</code>	117
5. Convertir ses données avec <code>std::stringstream</code>	118
6. Adieu, transtypages C style	119
12. Pointeurs et références	121
7. Démystifier les pointeurs	121
8. Qu'est-ce qu'une référence ?	122
9. Quelle différence entre les deux ?	122
10. Transmettre des paramètres par valeur ou par référence ?	124
Le type de l'objet passé en paramètre ?	124
L'espace mémoire nécessaire ?	124
La nécessité de répercuter les modifications ?	125
Disposer de comportements polymorphes ?	126
11. Transmettre des paramètres, par référence ou par pointeur ?	127
Pourquoi préférer le passage par référence ?	127
Les pointeurs si vous n'avez pas le choix	128
La non-existence d'une donnée	129
Renvoyer un objet polymorphe	130
12. J'ai entendu parler de... Rvalue-reference ?	133
13. La constance, autant que faire se peut.....	135
13. Et les fonctions membres ?	136
14. Pourquoi préférer la constance quand c'est possible ?	137
14. Comprendre le polymorphisme	139
15. RAll, le mal nommé	143
16. Préférez les pointeurs intelligents	145
17. Salade de sémantique	147
18. Coplien, règle des trois grands et compilateur.....	149
15. La règle des trois grands	150
16. Quand les trois ne font plus que deux	157
17. Quand les trois deviennent cinq	158
19. Et pour la sémantique d'entité ?.....	160
18. Interdire la copie et l'affectation	160
19. Ce qui change pour le destructeur	163

20. Interdire la destruction du type de base	164
21. Autoriser la destruction du type de base	167
22. Laquelle choisir ?.....	169
20. Du bon usage des opérateurs	170
23. Gardez la sémantique des opérateurs	170
24. Quelle forme leur donner ?.....	170
Étude de cas : Un jeu d'échecs en 3000 lignes de code.....	174
21. Les spécifications	175
22. Un jeu d'échecs, c'est quoi ?.....	176
1. Subdivisez les problèmes	176
2. Allez au fond des choses	177
3. Méfiez-vous des termes trop génériques	179
23. Les concepts de base	180
4. Les joueurs	180
5. L'échiquier : le champ de bataille.....	180
6. La case : une partie du champ de bataille.....	180
7. La couleur : pour distinguer les éléments.....	181
8. Se déplacer : quitter un endroit pour aller ailleurs.....	181
9. La pièce : un membre de l'équipe.....	182
10. Échec, mat et pat : des situations particulières.....	182
11. La disposition de départ	182
24. Les règles	184
12. Les règles permissives	185
13. Les règles restrictives	185
14. Et les exceptions	186
15. Le roque	187
16. La promotion du pion	187
25. Quelques systèmes connexes	188
17. Le système d'arbitrage	188
18. Un système d'introduction des commandes	189
19. Un système de représentation de la situation	190
20. Un système de commandes	190
26. Les évolutions probables	192
21. Respect des règles laissées de côté	193

22. Sauvegarde des données d'une partie	193
23. Mise en place d'un historique	194
24. Mise en place d'une <i>intelligence artificielle</i>	194
25. Une meilleure représentation visuelle	194
26. Des jeux et des parties alternatives	195
27. Analyse stratégique d'une partie	195
28. Des parties chronométrées	195
29. Jeu en réseau	196
30. Toutes les évolutions auxquelles on n'a pas pensé	196
27. De l'analyse des besoins à l'analyse technique.....	198
31. Un nom = un type, un verbe = une fonction	198
32. Un type, une fonction = une responsabilité	199
28. Les données métier	201
33. Le besoin d'éléments non copiables	201
34. La couleur	203
35. La coordonnée	205
36. La notion de déplacement	207
Les directions de base	209
37. La pièce de base	210
38. Les pièces concrètes	213
39. Les cases et l'échiquier	221
40. La fabrique, un besoin de développeur	225
41. Retour sur l'échiquier	232
42. Le visiteur, encore un besoin de développeur	233
43. Identifier le roi	237
29. Les fonctions d'aide	239
44. Convertir les coordonnées	239
45. Trouver le roi	242
46. Déterminer la direction d'un déplacement	243
47. Définir le nombre de cases traversées	244
48. Normaliser les chaînes de caractères	245
30. Respecter les règles	247
49. Implémenter les règles permissives	247
50. Les règles restrictives	252
51. Repérer les situations d'échec	256
31. Anticipation et simulation	259
52. Les index et la simulation	260

53. L'échiquier et la simulation	261
54. Simulation et situations d'échec	263
55. Tester tous les déplacements	272
32. Le joueur	275
33. Focus sur l'interactivité	277
56. N'importe quel système d'entrées	278
57. N'importe quel système de sorties	280
58. N'importe quelle commande	283
59. Envoyer une commande quelconque	284
60. Les entrées spécifiques au joueur	286
34. Faisons le point	289
61. Très bien, mais où cela nous mène-t-il ?.....	290
62. Commandes exécutables et commandes concrètes	294
35. Les commandes du mode console	296
63. La peste ou le choléra ?.....	296
64. Ni l'un ni l'autre : des politiques.....	300
65. La commande non reconnue	301
66. La commande d'aide	306
67. Le besoin d'une nouvelle partie	308
68. Quelques affichages spécifiques	310
69. Les commandes agissant sur la partie	315
70. Les commandes de déplacement	323
36. Émission des commandes	334
37. Les entrées/sorties en mode console.....	338
71. Le système d'entrées en mode console	338
72. L'affichage en mode console	343
73. Implémenter les affichages des commandes	347
38. La partie	352
39. La fonction <i>main()</i>	358
40. Le mot de la fin	359
74. Ce qu'il faut retenir	361
Annexe	362
41. Facilitez-vous la vie	363

1. La documentation à portée de main	363
2. Gardez l'historique	364
3. Traquez la bestiole	365
4. Automatisez la compilation	366
42. La compilation n'est pas une science occulte.....	369
5. Le préprocesseur : quelques commandes simples.....	370
6. Un programme pour écrire des programmes	371
L'analyse lexicale	371
L'analyse syntaxique	371
Les optimisations	372
7. Générer le code assembleur	373
8. L'assembleur	373
9. Un fichier à la fois	373
10. L'édition de liens	374
43. Le poids du passé	375
11. Le "bon vieux temps" des cartes perforées	375
12. SESE ? et pourquoi pas RAll ?.....	377
Liste des tableaux	380
Index	381