知識情報処理演習 I, II レポート

理工学研究科情報工学専攻情報工学コース 1 年
17NM708N 鈴木雅也

2017 年 10 月 25 日

1 はじめに

本レポートでは, ゲーム木探索に対する理解を目標に NegaMax 法を用いたオセロのプログラムを作成した.

2 開発環境

開発環境は表 2 のようになっている. ここで Kotlin[*1]とは JVM 上で動作するプログラミング言語であり, Java との互換性を有している.

表 1　開発環境

OS	Ubuntu 16.04 LTS
プログラミング言語	Kotlin 1.1
JDK	Java SE Development Kit 8
ライブラリ	JavaFX
IDE	IntelliJ IDEA Ultimate 2017.2
	Scene Builder 8.3.0

3 ディレクトリ構造

3.1 リソース

- settings.xml (ソースコード 1)
  プログラムの設定
- resources.xml (ソースコード 2)
  デフォルトのリソース
- resources_ja.xml (ソースコード 3)
  日本語用リソース

[*1] http://kotlinlang.org/

3.2 プログラム io/github/massongit/othello2017/kotlin/

- main.kt (ソースコード 4)
  main メソッド (プログラムの始点)

3.2.1 アプリケーション app/

- MainApplication.kt (ソースコード 5)
  GUI の始点 (画面の切り替え等を行う)
- DisplayType.kt (ソースコード 6)
  画面の種類
- メニュー画面 menu/
  - Menu.fxml (ソースコード 7)
    メニュー画面の View
  - Menu.css (ソースコード 8)
    メニュー画面用 CSS
  - MenuController.kt (ソースコード 9)
    メニュー画面の Controller
- スタート画面 start/
  - StartDisplay.fxml (ソースコード 10)
    スタート画面の View
  - StartDisplay.css (ソースコード 11)
    スタート画面用 CSS
  - StartDisplayController.kt (ソースコード 12)
    スタート画面の Controller
- プレイ画面 play/
  - Node.kt (ソースコード 13)
    ゲームの各ターンに対応したノードの雛形
  - PlayDisplay.fxml (ソースコード 14)
    プレイ画面の View
  - PlayDisplay.css (ソースコード 15)
    プレイ画面の CSS
  - PlayDisplayController.kt (ソースコード 16)
    プレイ画面の Controller. ゲームの各ターンに対応したノードの一種として扱われている.

- 情報パネル information/
  - Information.fxml (ソースコード 17)
    情報パネルの View
  - Information.css (ソースコード 18)
    情報パネルの CSS
  - InformationController.kt (ソースコード 19)
    情報パネルの Controller
- 石 stone/
  - Stone.kt (ソースコード 20)
    石を表す GUI 部品
  - Move.kt (ソースコード 21)
    着手を表す GUI 部品. 石の一種として扱われている.
  - StoneState.kt (ソースコード 22)
    石の種類
- AI ai/
  - AI.kt (ソースコード 23)
    AI の雛形
  - AIStrength.kt (ソースコード 24)
    AI の強さ
  - ノード node/
    - AINode.kt (ソースコード 25)
      AI によるゲーム木探索で使用するノードの雛形
    - AINodeType.kt (ソースコード 26)
      AI によるゲーム木探索で使用するノードの種類
  - NegaMax 法による AI nega_max/
    - NegaMaxAI.kt (ソースコード 27)
      NegaMax 法による AI
    - NegaMaxNode.kt (ソースコード 28)
      NegaMax 法におけるゲーム木探索で使用するノード

3.2.2 **ユーティリティ** utils/

- XMLResourceBundleControl.kt (ソースコード 29)
  XML 形式の設定ファイルに対応した Controller
- big_integer.kt (ソースコード 30)
  BigInteger のビット演算を Kotlin のビット演算子と同様に記述できるようにするためのライブラリ

4 実装

4.1 盤面処理

Wikipedia[1] を参考に, ビット演算による盤面処理の高速化を行った. この際, Kotlin の基本型のひとつである Long 型では盤面全体を表現し切れないため, ビットを扱うオブジェクトとして, Java の標準ライブラリである java.math.BigInteger を用いている. また, 盤面処理を着手の探索の際に行い, 反転処理ではその結果を用いることで反転処理の高速化を行っている.

4.2 AI

次の 2 種類の AI を実装した. なお. AI に人間らしさを出すため, AI の思考時間を実際の思考時間より 300〜1199ms 程度長く取っている.

- 強い AI
  ユーザがどんなに強くても勝ちに行くことを目指した AI. Negamax 法と β カットを用い, 10 手先まで読む. 葉ノードにおける着手の数を x, 葉ノードの子ノードにおける着手の数をそれぞれ $y_1, y_2, ..., y_n$ としたとき, 評価関数は次のようになる.

$$x + \max(-y_1, -y_2, ..., -y_n) \tag{1}$$

- 弱い AI
  ユーザがどんなに弱くても負けに行くことを目指した AI. 強い AI をベースに, 次のような変更を加えた.
  - 最大値を取る箇所で最小値を取る
  - β カットの代わりに α カットを用いる

5　スクリーンショット

図 1　スタート画面

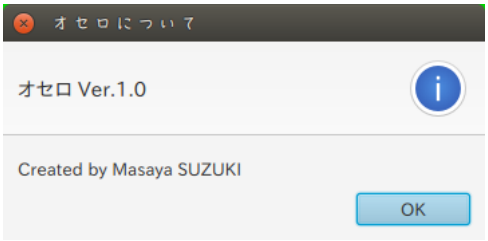

図 2　バージョン情報の表示

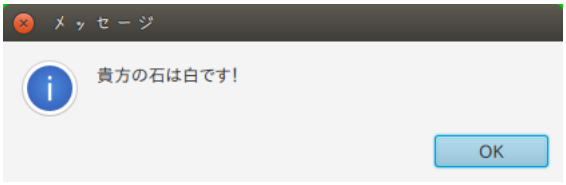

図 3　先手後手の表示

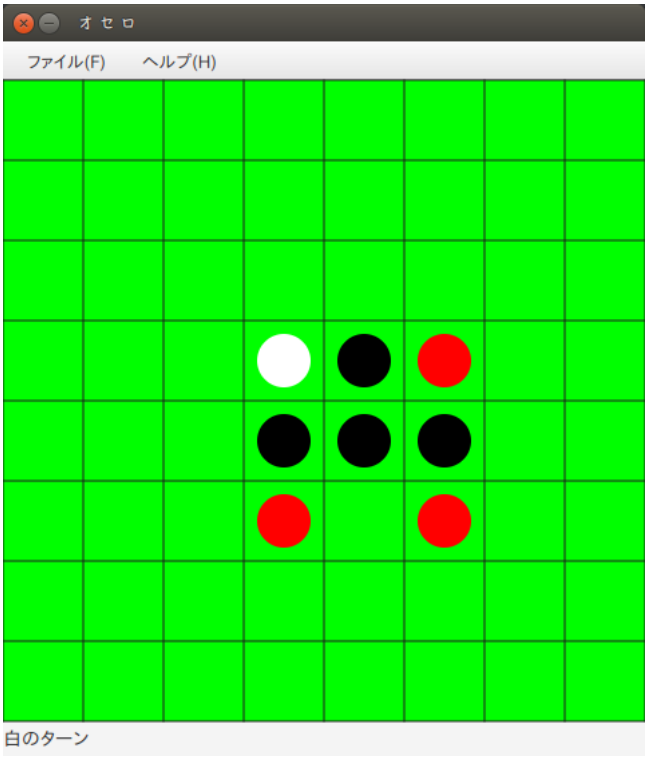

図 4　プレイ画面

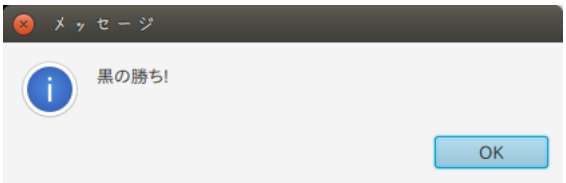

図 5　勝敗の表示

図 6　もう一度プレイするかどうかの選択

6 考察

強い AI においては, 一般に不利になると言われている端よりひとつ内側のマスをユーザに取らせようとする傾向があり, 弱い AI においては, 一般に有利になると言われている端のマスをユーザに取らせようとする傾向があった. このことから, AI が意図した通りの動作を行っていることがわかった.

7 まとめ

本レポートでは, ゲーム木探索に対する理解を目標に NegaMax 法を用いたオセロのプログラムを作成した. 実際の実装を通して, NegaMax 法や α カットや β カット, 盤面処理といったゲーム木探索で用いられるアルゴリズムやテクニックを理解することができたことから, ゲーム木探索に対する理解という目標は達成できたと考えられる.

参考文献

[1] オセロにおけるビットボード - wikipedia. https://ja.wikipedia.org/wiki/%E3%82%AA%E3%82%BB%E3%83%AD%E3%81%AB%E3%81%8A%E3%81%91%E3%82%8B%E3%83%93%E3%83%83%E3%83%88%E3%83%9C%E3%83%BC%E3%83%89.

A ソースコード

ソースコード 1 settings.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3  <properties>
4      <entry key="version">1.0</entry>
5  </properties>
```

ソースコード 2 resources.xml

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3  <properties>
4      <entry key="title">Othello</entry>
5      <entry key="start">Start</entry>
6      <entry key="strength-of-ai">Strength of AI</entry>
7      <entry key="strong-ai">Strong</entry>
8      <entry key="weak-ai">Weak</entry>
9      <entry key="file">_File</entry>
10     <entry key="reset">_Reset</entry>
11     <entry key="close">_Close</entry>
12     <entry key="help">_Help</entry>
```

```
13      <entry key="about">_About</entry>
14      <entry key="your-stone-is-black">Your stone is Black!</entry>
15      <entry key="your-stone-is-white">Your stone is White!</entry>
16      <entry key="play-first-tern">Black's tern</entry>
17      <entry key="draw-first-tern">White's tern</entry>
18      <entry key="error-dialog-title">Error</entry>
19      <entry key="error-dialog-label">The exception stacktrace was:</entry>
20      <entry key="play-first-win">Black Win!</entry>
21      <entry key="draw-first-win">White Win!</entry>
22      <entry key="draw">Draw!</entry>
23      <entry key="replay">Do you play this game again?</entry>
24      <entry key="credit">Created by Masaya SUZUKI</entry>
25  </properties>
```

ソースコード 3　resources_ja.xml

```
1   <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2   <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3   <properties>
4       <entry key="title">オセロ</entry>
5       <entry key="start">スタート</entry>
6       <entry key="strength-of-ai">AIの強さ</entry>
7       <entry key="strong-ai">強い</entry>
8       <entry key="weak-ai">弱い</entry>
9       <entry key="file">ファイル(_F)</entry>
10      <entry key="reset">リセット(_N)</entry>
11      <entry key="close">閉じる(_X)</entry>
12      <entry key="help">ヘルプ(_H)</entry>
13      <entry key="about">オセロについて</entry>
14      <entry key="your-stone-is-black">貴方の石は黒です！</entry>
15      <entry key="your-stone-is-white">貴方の石は白です！</entry>
16      <entry key="play-first-tern">黒のターン</entry>
17      <entry key="draw-first-tern">白のターン</entry>
18      <entry key="error-dialog-title">エラー</entry>
19      <entry key="error-dialog-label">スタックトレース:</entry>
20      <entry key="play-first-win">黒の勝ち!</entry>
21      <entry key="draw-first-win">白の勝ち!</entry>
22      <entry key="draw">引き分け！</entry>
23      <entry key="replay">もう一度プレイしますか？</entry>
24      <entry key="credit">Created by Masaya SUZUKI</entry>
25  </properties>
```

ソースコード 4　io/github/massongit/othello2017/kotlin/main.kt

```
1   package io.github.massongit.othello2017.kotlin
2
3   import io.github.massongit.othello2017.kotlin.app.MainApplication
4   import javafx.application.Application
5
```

```
6  /**
7   * エントリポイント
8   * ( https://github.com/seraphy/JavaFXSimpleApp/blob/master/src/jp/seraphyware/
        simpleapp6/SimpleApp6.java を元に作成)
9   * @param args プログラム引数
10  * @author seraphy, Masaya SUZUKI
11  */
12 fun main(args: Array<String>) = Application.launch(MainApplication::class.java, *
       args)
```

ソースコード 5　io.github.massongit.othello2017.kotlin.app.MainApplication

```
1  package io.github.massongit.othello2017.kotlin.app
2
3  import io.github.massongit.othello2017.kotlin.app.play.ai.AIStrength
4  import io.github.massongit.othello2017.kotlin.utils.XMLResourceBundleControl
5  import javafx.application.Application
6  import javafx.fxml.FXMLLoader
7  import javafx.scene.Scene
8  import javafx.scene.control.Alert
9  import javafx.scene.control.Alert.AlertType
10 import javafx.scene.control.Label
11 import javafx.scene.control.TextArea
12 import javafx.scene.layout.GridPane
13 import javafx.scene.layout.Priority
14 import javafx.stage.Stage
15 import java.io.FileInputStream
16 import java.io.PrintWriter
17 import java.io.StringWriter
18 import java.nio.file.Paths
19 import java.util.*
20
21 /**
22  * メインアプリケーション
23  * ( https://github.com/seraphy/JavaFXSimpleApp/blob/master/src/jp/seraphyware/
        utils/ErrorDialogUtils.java を元に作成)
24  * @author seraphy, Masaya SUZUKI
25  */
26 class MainApplication : Application() {
27     companion object {
28         /**
29          * ステージ
30          */
31         lateinit var stage: Stage
32
33         /**
34          * AIの強さ
35          */
36         var aiStrength: AIStrength = AIStrength.STRONG
```

```
37
38          /**
39           * プロパティ
40           */
41          val properties: Properties = Properties().apply { loadFromXML(
                FileInputStream(Paths.get(MainApplication::class.java.protectionDomain
                .codeSource.location.toURI()).resolve("settings.xml").toString())) }
42
43          /**
44           * リソースバンドル
45           */
46          private val RESOURCES: ResourceBundle = ResourceBundle.getBundle("
                resources", XMLResourceBundleControl())
47
48          /**
49           * 画面を遷移させる
50           * @param display 遷移先の画面
51           */
52          fun translateDisplay(display: DisplayType) {
53              stage.scene = Scene(FXMLLoader(MainApplication::class.java.
                    getResource(Paths.get(MainApplication::class.qualifiedName?.
                    replace(".", "/")).parent.relativize(display.fxmlPath).toString
                    ()), RESOURCES).load())
54              stage.apply {
55                  // ステージの表示
56                  show()
57
58                  // リサイズできないようにする
59                  maxWidth = width
60                  minWidth = width
61                  maxHeight = height
62                  minHeight = height
63              }
64          }
65      }
66
67      override fun start(primaryStage: Stage) {
68          try {
69              stage = primaryStage.apply { title = RESOURCES.getString("title") }
70
71              // スタート画面へ遷移する
72              translateDisplay(DisplayType.START)
73          } catch (ex: Exception) {
74              // スタックトレースを標準エラー出力へ出力する
75              ex.printStackTrace()
76
77              // エラーダイアログ
78              val errorDialog = Alert(AlertType.ERROR, ex.message).apply {
79                  title = RESOURCES.getString("error-dialog-title")
```

```
80                  headerText = ex::class.simpleName
81              }
82
83              // スタックトレース
84              val stackTrace = StringWriter()
85              PrintWriter(stackTrace).use {
86                  ex.printStackTrace(it)
87              }
88
89              // テキストエリア
90              val textArea = TextArea(stackTrace.toString()).apply {
91                  isEditable = false
92                  isWrapText = true
93                  maxWidth = Double.MAX_VALUE
94                  maxHeight = Double.MAX_VALUE
95              }
96              GridPane.setVgrow(textArea, Priority.ALWAYS)
97              GridPane.setHgrow(textArea, Priority.ALWAYS)
98
99              // 詳細コンテンツをセット
100             errorDialog.dialogPane.expandableContent = GridPane().apply {
101                 maxWidth = Double.MAX_VALUE
102                 add(Label(RESOURCES.getString("error-dialog-label")), 0, 0)
103                 add(textArea, 0, 1)
104             }
105
106             // エラーダイアログを表示
107             errorDialog.showAndWait()
108         }
109     }
110 }
```

ソースコード 6　io.github.massongit.othello2017.kotlin.app.DisplayType

```
1  package io.github.massongit.othello2017.kotlin.app
2
3  import java.nio.file.Path
4  import java.nio.file.Paths
5
6  /**
7   * 画面の種類
8   * @author Masaya SUZUKI
9   */
10 enum class DisplayType {
11     /**
12      * スタート画面
13      */
14     START {
```

```
15          override val fxmlPath: Path = super.fxmlPath.resolve("start").resolve("
                StartDisplay.fxml")
16      },
17
18      /**
19       * プレイ画面
20       */
21      PLAY {
22          override val fxmlPath: Path = super.fxmlPath.resolve("play").resolve("
                PlayDisplay.fxml")
23      };
24
25      /**
26       * 画面のFXMLのパス
27       */
28      open val fxmlPath: Path = Paths.get(DisplayType::class.qualifiedName?.replace
            (".", "/")).parent
29  }
```

ソースコード 7 io/github/massongit/othello2017/kotlin/app/menu/Menu.fxml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.scene.control.*?>
4  <?import javafx.scene.input.KeyCodeCombination?>
5  <MenuBar xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml" id="
       menu-bar"
6            stylesheets="@Menu.css" fx:controller="io.github.massongit.othello2017.
                 kotlin.app.menu.MenuController">
7      <Menu text="%file">
8          <MenuItem onAction="#onReset" text="%reset">
9              <accelerator>
10                 <KeyCodeCombination alt="UP" code="N" control="UP" meta="UP"
                       shift="UP" shortcut="DOWN"/>
11             </accelerator>
12         </MenuItem>
13         <MenuItem onAction="#onClose" text="%close">
14             <accelerator>
15                 <KeyCodeCombination alt="UP" code="Q" control="UP" meta="UP"
                       shift="UP" shortcut="DOWN"/>
16             </accelerator>
17         </MenuItem>
18     </Menu>
19     <Menu text="%help">
20         <MenuItem onAction="#onAbout" text="%about"/>
21     </Menu>
22 </MenuBar>
```

ソースコード 8　io/github/massongit/othello2017/kotlin/app/menu/Menu.css

```
/*メニューバー*/
#menu-bar {
    -fx-use-system-menu-bar: true;
}
```

ソースコード 9　io.github.massongit.othello2017.kotlin.app.menu.MenuController

```
package io.github.massongit.othello2017.kotlin.app.menu

import io.github.massongit.othello2017.kotlin.app.DisplayType
import io.github.massongit.othello2017.kotlin.app.MainApplication
import javafx.application.Platform
import javafx.fxml.FXML
import javafx.fxml.Initializable
import javafx.scene.control.Alert
import javafx.scene.control.Alert.AlertType
import java.net.URL
import java.util.*


/**
 * メニューコントローラ
 * @author Masaya SUZUKI
 */
class MenuController : Initializable {
    /**
     * リソースバンドル
     */
    private lateinit var resources: ResourceBundle

    override fun initialize(location: URL, resources: ResourceBundle) {
        this.resources = resources
    }

    /**
     * ゲームをリセットする
     */
    @FXML
    fun onReset() = MainApplication.translateDisplay(DisplayType.START)

    /**
     * プログラムを終了する
     */
    @FXML
    fun onClose() = Platform.exit()

    /**
     * バージョン情報を表示する
```

```
42          */
43      @FXML
44      private fun onAbout() = Alert(AlertType.INFORMATION, this.resources.getString
            ("credit")).apply {
45          title = resources.getString("about")
46          headerText = "${resources.getString("title")} Ver.${MainApplication.
                properties.getProperty("version")}"
47      }.show()
48  }
```

ソースコード 10　io/github/massongit/othello2017/kotlin/app/start/StartDisplay.fxml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3   <?import javafx.geometry.Insets?>
4   <?import javafx.scene.control.*?>
5   <?import javafx.scene.layout.*?>
6   <?import javafx.scene.text.Font?>
7   <VBox prefHeight="563.0" prefWidth="480.0" stylesheets="@StartDisplay.css, @../
        play/PlayDisplay.css"
8         xmlns="http://javafx.com/javafx/8.0.121" xmlns:fx="http://javafx.com/fxml/1
              "
9         fx:controller="io.github.massongit.othello2017.kotlin.app.start.
              StartDisplayController">
10      <fx:include source="../menu/Menu.fxml"/>
11      <VBox alignment="CENTER" fillWidth="false" prefHeight="506.0" prefWidth="
            480.0" styleClass="board">
12          <Label id="title" alignment="CENTER" prefHeight="60.0" prefWidth="180.0"
                text="%title">
13              <font>
14                  <Font size="43.0"/>
15              </font>
16              <VBox.margin>
17                  <Insets bottom="6.0"/>
18              </VBox.margin>
19          </Label>
20          <TitledPane animated="false" text="%strength-of-ai">
21              <VBox prefHeight="56.0" prefWidth="0.0">
22                  <RadioButton fx:id="strongAI" mnemonicParsing="false" selected="
                        true" text="%strong-ai">
23                      <toggleGroup>
24                          <ToggleGroup fx:id="aiStrength"/>
25                      </toggleGroup>
26                  </RadioButton>
27                  <RadioButton fx:id="weakAI" mnemonicParsing="false" text="%weak-
                        ai" toggleGroup="$aiStrength"/>
28              </VBox>
29              <padding>
30                  <Insets bottom="12.0" top="12.0"/>
```

```
31                  </padding>
32              </TitledPane>
33              <Button mnemonicParsing="false" onMouseClicked="#onClick" text="%start">
34                  <VBox.margin>
35                      <Insets/>
36                  </VBox.margin>
37              </Button>
38          </VBox>
39      </VBox>
```

ソースコード 11　io/github/massongit/othello2017/kotlin/app/start/StartDisplay.css

```
1  /*タイトル*/
2  #title {
3      -fx-background-color: white;
4  }
```

ソースコード 12　io.github.massongit.othello2017.kotlin.app.start.StartDisplayController

```
1  package io.github.massongit.othello2017.kotlin.app.start
2
3  import io.github.massongit.othello2017.kotlin.app.DisplayType
4  import io.github.massongit.othello2017.kotlin.app.MainApplication
5  import io.github.massongit.othello2017.kotlin.app.play.ai.AIStrength
6  import javafx.fxml.FXML
7  import javafx.fxml.Initializable
8  import javafx.scene.control.ToggleButton
9  import javafx.scene.control.ToggleGroup
10 import java.net.URL
11 import java.util.*
12
13 /**
14  * スタート画面コントローラ
15  * @author Masaya SUZUKI
16  */
17 class StartDisplayController : Initializable {
18     /**
19      * AIの強さ
20      */
21     @FXML
22     private lateinit var aiStrength: ToggleGroup
23
24     /**
25      * 強いAI
26      */
27     @FXML
28     private lateinit var strongAI: ToggleButton
29
30     /**
```

```
31          * 弱いAI
32          */
33         @FXML
34         private lateinit var weakAI: ToggleButton
35
36         override fun initialize(location: URL, resources: ResourceBundle) {
37             this.strongAI.userData = AIStrength.STRONG
38             this.weakAI.userData = AIStrength.WEAK
39         }
40
41         /**
42          * スタートボタンをクリックしたときのイベント
43          */
44         @FXML
45         private fun onClick() {
46             // AIの強さをセットする
47             MainApplication.aiStrength = this.aiStrength.selectedToggle.userData as
                   AIStrength
48
49             // プレイ画面へ遷移する
50             MainApplication.translateDisplay(DisplayType.PLAY)
51         }
52     }
```

ソースコード 13　io.github.massongit.othello2017.kotlin.app.play.Node

```
1   package io.github.massongit.othello2017.kotlin.app.play
2
3   import io.github.massongit.othello2017.kotlin.app.play.stone.Move
4   import io.github.massongit.othello2017.kotlin.app.play.stone.StoneState
5   import io.github.massongit.othello2017.kotlin.utils.*
6   import javafx.scene.layout.GridPane
7   import java.math.BigInteger
8
9   /**
10   * ノード
11   * (https://ja.wikipedia.org/wiki/%E3%82%AA%E3%82%BB%E3%83%AD%E3%81%AB%E3%81%8A%
         E3%81%91%E3%82%8B%E3%83%93%E3%83%83%E3%83%88%E3%83%9C%E3%83%BC%E3%83%89 を元
         に作成)
12   * @author Masaya SUZUKI
13   */
14  abstract class Node<out T> {
15      /**
16       * 現在のターン
17       */
18      internal open var tern: StoneState = StoneState.PLAY_FIRST
19
20      /**
21       * 着手のリスト
```

```
22          */
23      internal open val moveList: MutableList<Move> = mutableListOf()
24
25      /**
26       * ビットボード
27       */
28      internal open val bitBoard: MutableMap<StoneState, BigInteger> = mutableMapOf
            (
29              StoneState.PLAY_FIRST to BigInteger.ZERO, // 先手
30              StoneState.DRAW_FIRST to BigInteger.ZERO // 後手
31      )
32
33      /**
34       * 盤面のマスの数
35       */
36      private val BOARD_SIZE: Int = 64
37
38      /**
39       * 石を追加する
40       * @param columnIndex 列
41       * @param rowIndex 行
42       * @param stoneState 石の種類
43       */
44      internal open fun addStone(columnIndex: Int, rowIndex: Int, stoneState:
            StoneState) {
45          this.bitBoard[stoneState] = this.bitBoard[stoneState]!! or this.
                getStoneBit(columnIndex, rowIndex)
46      }
47
48      /**
49       * 次のターンへ移る
50       * @param stoneState 次のターンの先手・後手
51       * @return 次のターンのノードのリスト
52       */
53      internal open fun nextTern(stoneState: StoneState): List<T> {
54          // 自分のターン -> 相手のターンの順に次のターンへ移れるかどうかを見る
55          for (state in listOf(stoneState, stoneState.inv())) {
56              val result = this.nextTernPerState(state)
57              if (result != null) {
58                  return result
59              }
60          }
61
62          // どのターンにも移れない場合、ゲームを終了する
63          this.endGame()
64
65          return listOf()
66      }
67
```

```
68      /**
69       * 次のターンへ移る
70       * @param state 石
71       * @return 次のターンのノードのリスト
72       */
73      internal open fun nextTernPerState(state: StoneState): List<T>? {
74          // ビットボードを動かすラムダ式のリスト
75          val transfers: List<(BigInteger) -> BigInteger> = listOf<(BigInteger) ->
                BigInteger>(
76              { (it shl 1) and BigInteger("fefefefefefefe", 16) }, // 左方向へ1マス
77              { (it ushr 1) and BigInteger("7f7f7f7f7f7f7f7f", 16) }, // 右方向へ1マス
78              { (it shl 7) and BigInteger("7f7f7f7f7f7f7f00", 16) }, // 右上方向へ1マス
79              { (it ushr 7) and BigInteger("fefefefefefefe", 16) }, // 左下方向へ1マス
80              { (it shl 8) and BigInteger("ffffffffffffff00", 16) }, // 上方向へ1マス
81              { (it ushr 8) and BigInteger("ffffffffffffff", 16) }, // 下方向へ1マス
82              { (it shl 9) and BigInteger("fefefefefefefe00", 16) }, // 左上方向へ1マス
83              { (it ushr 9) and BigInteger("7f7f7f7f7f7f7f7f", 16) } // 右下方向へ1マス
84          )
85
86          // 着手のビットボード
87          var movePattern = BigInteger.ZERO
88
89          // 石が置けそうな場所を探す
90          for (transfer in transfers) {
91              movePattern = movePattern or transfer(this.bitBoard[state.inv()]!!)
92          }
93          movePattern = movePattern and (this.bitBoard[state]!! or this.bitBoard[
                state.inv()]!!).inv()
94
95          // 実際に石が置ける場所が見つかったかどうか
96          var isFindMove = false
97
98          // マスク
99          var mask = BigInteger.ONE
100
101         // 実際に石が置ける場所を探し、石を置いていく
102         for (i in this.BOARD_SIZE - 1 downTo 0) {
103             if (movePattern == BigInteger.ZERO) { // 石を置けそうな場所がない場合、ループを抜ける
104                 break
105             } else {
```

```
106                 // 現在見ている位置
107                 val currentPosition = movePattern and mask
108
109                 // 現在見ている位置が石を置ける場所の候補に入っているとき
110                 if (currentPosition != BigInteger.ZERO) {
111                     // 反転する石が立ったビットボード
112                     var reversePattern = BigInteger.ZERO
113
114                     // 各方向について、反転する石のビットを立てる
115                     for (transfer in transfers) {
116                         // 現在見ている方向で反転する石が立ったビットボード
117                         var reverseStones = BigInteger.ZERO
118
119                         // マスク
120                         var reverseStoneMask = currentPosition
121
122                         // 現在見ている方向へ石を反転させていく
123                         while (true) {
124                             reverseStoneMask = transfer(reverseStoneMask)
125                             if (reverseStoneMask and this.bitBoard[state.inv()]!!
                                    == BigInteger.ZERO) { // 反転させられない場所にきた場合、ループを抜ける
126                                 break
127                             } else {
128                                 reverseStones = reverseStones or reverseStoneMask
129                             }
130                         }
131
132                         // 石を反転させていき、最後に自分の石に到達した場合、反転する石が立ったビットボードへ結果をマージ
133                         if (reverseStoneMask and this.bitBoard[state]!! !=
                                BigInteger.ZERO) {
134                             reversePattern = reversePattern or reverseStones
135                         }
136                     }
137
138                     if (reversePattern != BigInteger.ZERO) { // 実際に石が置ける場所に石を置く
139                         // 着手を追加する
140                         this.addMove(Move(reversePattern), this.
                                getStoneColumnIndex(i), this.getStoneRowIndex(i))
141
142                         isFindMove = true
143                     }
144
145                     movePattern = movePattern xor mask
146                 }
147
148                 mask = mask shl 1
```

```
149                 }
150             }
151
152             if (isFindMove) { // 実際に石が置けるとき
153                 return this.changeTern(state)
154             } else { // 石が置けないとき
155                 return null
156             }
157         }
158
159         /**
160          * 着手を追加する
161          * @param move 着手
162          * @param columnIndex 列
163          * @param rowIndex 行
164          */
165         internal open fun addMove(move: Move, columnIndex: Int, rowIndex: Int) {
166             // 着手の座標をセット
167             GridPane.setColumnIndex(move, columnIndex)
168             GridPane.setRowIndex(move, rowIndex)
169
170             // リストに着手を追加する
171             this.moveList.add(move)
172         }
173
174         /**
175          * ターンを進める
176          * @param state 石
177          * @return 次のターンのノードのリスト
178          */
179         internal open fun changeTern(state: StoneState): List<T> = listOf()
180
181         /**
182          * ターンを実行する
183          * @param move 着手
184          * @return ターンを実行することで得られたノードのリスト
185          */
186         internal fun executeTern(move: Move): List<T> = this.executeTern(move,
                GridPane.getColumnIndex(move), GridPane.getRowIndex(move))
187
188         /**
189          * ターンを実行する
190          * @param move 着手
191          * @param columnIndex 列
192          * @param rowIndex 行
193          * @return ターンを実行することで得られたノードのリスト
194          */
195         internal open fun executeTern(move: Move, columnIndex: Int, rowIndex: Int):
                List<T> {
```

```
196             // 反転処理前の自分のビットボード
197             val prevMyBitBoard = this.bitBoard[this.tern]!!
198 
199             // ビットボードに対して反転処理を行う
200             this.bitBoard[this.tern] = this.bitBoard[this.tern]!! or this.getStoneBit
                    (columnIndex, rowIndex) or move.reversePattern
201             this.bitBoard[this.tern.inv()] = this.bitBoard[this.tern.inv()]!! xor
                    move.reversePattern
202 
203             // 反転処理前後での自分のビットボードの差分
204             var myBitBoardDiff = prevMyBitBoard xor this.bitBoard[this.tern]!!
205 
206             // インスタンスボードに対して反転処理を行う
207             for (i in this.BOARD_SIZE - 1 downTo 0) {
208                 if (myBitBoardDiff == BigInteger.ZERO) {
209                     break
210                 } else {
211                     if (myBitBoardDiff and BigInteger.ONE == BigInteger.ONE) {
212                         this.addStone(this.getStoneColumnIndex(i), this.
                                getStoneRowIndex(i), this.tern)
213                     }
214                     myBitBoardDiff = myBitBoardDiff ushr 1
215                 }
216             }
217 
218             // 次のターンへ移る
219             return this.nextTern(this.tern.inv())
220         }
221 
222         /**
223          * ゲームを終了する
224          */
225         internal abstract fun endGame()
226 
227         /**
228          * 石の座標をボード上のインデックスに変換する
229          * @param columnIndex 列
230          * @param rowIndex 行
231          * @return ボード上のインデックス
232          */
233         internal fun getStoneBoardIndex(columnIndex: Int, rowIndex: Int): Int = 8 *
                rowIndex + columnIndex
234 
235         /**
236          * ボード上のインデックスから列を取得する
237          * @param index ボード上のインデックス
238          * @return 列
239          */
240         private fun getStoneColumnIndex(index: Int): Int = index % 8
```

```
241
242         /**
243          * ボード上のインデックスから行を取得する
244          * @param index ボード上のインデックス
245          * @return 行
246          */
247         private fun getStoneRowIndex(index: Int): Int = index / 8
248
249         /**
250          * 石の座標を表すビットのみが立ったビットボードを取得する
251          * @param columnIndex 列
252          * @param rowIndex 行
253          * @return 石の座標を表すビットのみが立ったビットボード
254          */
255         private fun getStoneBit(columnIndex: Int, rowIndex: Int): BigInteger =
                 BigInteger.ONE shl (this.BOARD_SIZE - this.getStoneBoardIndex(columnIndex,
                  rowIndex) - 1)
256 }
```

ソースコード 14 io/github/massongit/othello2017/kotlin/app/play/PlayDisplay.fxml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.scene.layout.*?>
4  <VBox stylesheets="@PlayDisplay.css" xmlns="http://javafx.com/javafx/8.0.112"
       xmlns:fx="http://javafx.com/fxml/1"
5          fx:controller="io.github.massongit.othello2017.kotlin.app.play.
               PlayDisplayController">
6      <fx:include fx:id="menu" source="../menu/Menu.fxml"/>
7      <GridPane fx:id="board" styleClass="board">
8          <columnConstraints>
9              <ColumnConstraints halignment="CENTER" prefWidth="60.0"/>
10             <ColumnConstraints halignment="CENTER" prefWidth="60.0"/>
11             <ColumnConstraints halignment="CENTER" prefWidth="60.0"/>
12             <ColumnConstraints halignment="CENTER" prefWidth="60.0"/>
13             <ColumnConstraints halignment="CENTER" prefWidth="60.0"/>
14             <ColumnConstraints halignment="CENTER" prefWidth="60.0"/>
15             <ColumnConstraints halignment="CENTER" prefWidth="60.0"/>
16             <ColumnConstraints halignment="CENTER" prefWidth="60.0"/>
17         </columnConstraints>
18         <rowConstraints>
19             <RowConstraints prefHeight="60.0" valignment="CENTER"/>
20             <RowConstraints prefHeight="60.0" valignment="CENTER"/>
21             <RowConstraints prefHeight="60.0" valignment="CENTER"/>
22             <RowConstraints prefHeight="60.0" valignment="CENTER"/>
23             <RowConstraints prefHeight="60.0" valignment="CENTER"/>
24             <RowConstraints prefHeight="60.0" valignment="CENTER"/>
25             <RowConstraints prefHeight="60.0" valignment="CENTER"/>
26             <RowConstraints prefHeight="60.0" valignment="CENTER"/>
```

```
27            </rowConstraints>
28        </GridPane>
29        <fx:include fx:id="information" source="information/Information.fxml"/>
30    </VBox>
```

ソースコード 15　io/github/massongit/othello2017/kotlin/app/play/PlayDisplay.css

```
1   /*盤面*/
2   .board {
3       -fx-background-color: lime;
4   }
5
6   /*盤面*/
7   #board {
8       -fx-grid-lines-visible: true;
9   }
10
11  /*AIが思考している際の盤面*/
12  .board-for-ai {
13      -fx-opacity: 50%;
14  }
15
16  /*先手（黒）の石*/
17  .play-first-tern {
18      -fx-fill: black;
19  }
20
21  /*後手（白）の石*/
22  .draw-first-tern {
23      -fx-fill: white;
24  }
25
26  /*着手*/
27  .move {
28      -fx-fill: red;
29  }
30
31  /*ユーザ向け着手*/
32  .move-for-user {
33      -fx-cursor: hand;
34  }
```

ソースコード 16　io.github.massongit.othello2017.kotlin.app.play.PlayDisplayController

```
1   package io.github.massongit.othello2017.kotlin.app.play
2
3   import io.github.massongit.othello2017.kotlin.app.MainApplication
4   import io.github.massongit.othello2017.kotlin.app.menu.MenuController
5   import io.github.massongit.othello2017.kotlin.app.play.ai.AI
```

```
 6 import io.github.massongit.othello2017.kotlin.app.play.ai.nega_max.NegaMaxAI
 7 import io.github.massongit.othello2017.kotlin.app.play.information.
       InformationController
 8 import io.github.massongit.othello2017.kotlin.app.play.stone.Move
 9 import io.github.massongit.othello2017.kotlin.app.play.stone.Stone
10 import io.github.massongit.othello2017.kotlin.app.play.stone.StoneState
11 import javafx.animation.KeyFrame
12 import javafx.animation.Timeline
13 import javafx.application.Platform
14 import javafx.event.EventHandler
15 import javafx.fxml.FXML
16 import javafx.fxml.Initializable
17 import javafx.scene.control.Alert
18 import javafx.scene.control.Alert.AlertType
19 import javafx.scene.control.ButtonType
20 import javafx.scene.layout.GridPane
21 import javafx.util.Duration
22 import java.net.URL
23 import java.util.*
24
25 /**
26  * プレイ画面コントローラ
27  * @author Masaya SUZUKI
28  */
29 class PlayDisplayController : Initializable, Node<PlayDisplayController>() {
30     /**
31      * 盤面本体
32      */
33     @FXML
34     private lateinit var board: GridPane
35
36     /**
37      * メニューコントローラ
38      */
39     @FXML
40     private lateinit var menuController: MenuController
41
42     /**
43      * 情報ラベルコントローラ
44      */
45     @FXML
46     private lateinit var informationController: InformationController
47
48     /**
49      * リソースバンドル
50      */
51     private lateinit var resources: ResourceBundle
52
53     /**
```

```
54          * AIが担当するターン
55          */
56         private lateinit var aiTern: StoneState
57
58         /**
59          * AI
60          */
61         private val ai: AI = NegaMaxAI(MainApplication.aiStrength)
62
63         /**
64          * 乱数生成器
65          */
66         private val random: Random = Random()
67
68         /**
69          * インスタンスボード
70          */
71         private val instanceBoard: MutableMap<Int, Stone> = mutableMapOf()
72
73         /**
74          * AIが思考している際の盤面のスタイルクラス
75          */
76         private val BOARD_FOR_AI_STYLE_CLASS: String = "board-for-ai"
77
78         /**
79          * 出力用の列表記の対応表
80          */
81         private val COLUMN_INDEX_CORRESPONDENCE_TABLE = listOf("a", "b", "c", "d", "e
               ", "f", "g", "h")
82
83         override fun initialize(location: URL, resources: ResourceBundle) {
84             this.resources = resources
85
86             // 初期位置に石を置く
87             this.addStone(3, 4, StoneState.PLAY_FIRST)
88             this.addStone(4, 3, StoneState.PLAY_FIRST)
89             this.addStone(3, 3, StoneState.DRAW_FIRST)
90             this.addStone(4, 4, StoneState.DRAW_FIRST)
91
92             // AIのターンを決める
93             if (this.random.nextBoolean()) {
94                 this.aiTern = StoneState.PLAY_FIRST
95             } else {
96                 this.aiTern = StoneState.DRAW_FIRST
97             }
98
99             // プレイヤーが先手か後手かを表示する
100            Alert(AlertType.INFORMATION, this.resources.getString(this.aiTern.inv().
                   decideTernKey)).apply { headerText = null }.showAndWait()
```

```
101
102         // 先手のターンを始める
103         this.nextTern(this.tern)
104     }
105
106     override fun addStone(columnIndex: Int, rowIndex: Int, stoneState: StoneState
            ) {
107         // インデックス
108         val index = this.getStoneBoardIndex(columnIndex, rowIndex)
109
110         // 既に置かれている石を削除
111         this.removeStoneFromBoard(this.instanceBoard.remove(index))
112
113         // 石
114         val stone = Stone(stoneState)
115
116         // 盤面本体に石を追加する
117         this.board.add(stone, columnIndex, rowIndex)
118
119         // インスタンスボードに石を追加する
120         this.instanceBoard[index] = stone
121
122         super.addStone(columnIndex, rowIndex, stoneState)
123     }
124
125     override fun nextTern(stoneState: StoneState): List<PlayDisplayController> {
126         // 着手を全て削除する
127         for (i in 0 until this.moveList.size) {
128             this.removeStoneFromBoard(this.moveList.removeAt(0))
129         }
130
131         return super.nextTern(stoneState)
132     }
133
134     override fun endGame() = Platform.runLater {
135         // ダイアログ
136         val dialog = Alert(AlertType.INFORMATION).apply { headerText = null }
137
138         // 勝敗を出力
139         if (this.bitBoard[StoneState.PLAY_FIRST]!!.bitCount() == this.bitBoard[
                StoneState.DRAW_FIRST]!!.bitCount()) {
140             dialog.contentText = this.resources.getString("draw")
141             println("Draw")
142         } else {
143             val winStone: StoneState
144
145             if (this.bitBoard[StoneState.PLAY_FIRST]!!.bitCount() < this.bitBoard
                    [StoneState.DRAW_FIRST]!!.bitCount()) {
146                 winStone = StoneState.DRAW_FIRST
```

```
147                 } else {
148                     winStone = StoneState.PLAY_FIRST
149                 }
150
151                 dialog.contentText = this.resources.getString(winStone.winKey)
152                 println("Win: ${winStone.name}")
153             }
154
155             // ダイアログを表示
156             dialog.showAndWait()
157
158             if (Alert(AlertType.CONFIRMATION, this.resources.getString("replay")).
                    apply {
159                 headerText = null
160                 buttonTypes.setAll(ButtonType.YES, ButtonType.NO)
161             }.showAndWait().get() == ButtonType.YES) { // ゲームをもう一度プレイする
                    場合
162                 this.menuController.onReset()
163             } else { // ゲームを終了する場合
164                 this.menuController.onClose()
165             }
166         }
167
168         override fun addMove(move: Move, columnIndex: Int, rowIndex: Int) {
169             // 盤面本体に着手を追加する
170             this.board.children.add(move)
171
172             super.addMove(move, columnIndex, rowIndex)
173         }
174
175         override fun changeTern(state: StoneState): List<PlayDisplayController> {
176             // 現在見ているターンと実際のターンが一致しない場合、ターンを進める
177             if (state != this.tern) {
178                 this.tern = this.tern.inv()
179                 this.informationController.change(this.tern.styleClass)
180             }
181
182             if (this.tern == aiTern) { // AIのターンのとき
183                 // 盤面のスタイルクラスをAIが思考している際のものに変更する
184                 this.board.styleClass.add(this.BOARD_FOR_AI_STYLE_CLASS)
185
186                 // AIのターンを実行する
187                 // (思考時間 = 実際の思考時間 + 300〜1199ms)
188                 Timeline(KeyFrame(Duration.millis(300.0 + this.random.nextInt(900)),
                        EventHandler {
189                     // 盤面のスタイルクラスを通常のものに戻す
190                     this.board.styleClass.remove(this.BOARD_FOR_AI_STYLE_CLASS)
191
192                     // AIのターンをを実行する
```

```
193                     this.executeTern(this.ai.execute(this.tern, this.bitBoard.
                            toMutableMap(), this.moveList.toList()))
194                 })).play()
195             } else { // ユーザのターンの場合、着手をユーザ仕様に変更する
196                 for (move in this.moveList) {
197                     move.toUser(EventHandler { this.executeTern(it.source as Move) })
198                 }
199             }
200
201             return super.changeTern(state)
202         }
203
204         override fun executeTern(move: Move, columnIndex: Int, rowIndex: Int): List<
                PlayDisplayController> {
205             // 石の座標を標準出力へ出力
206             println("Put: ${this.COLUMN_INDEX_CORRESPONDENCE_TABLE[columnIndex]}${
                    rowIndex + 1} (${this.tern.name})")
207
208             return super.executeTern(move, columnIndex, rowIndex)
209         }
210
211         /**
212          * 盤面本体から石を削除する
213          * @param stone 石
214          */
215         private fun removeStoneFromBoard(stone: Stone?) = this.board.children.remove(
                stone)
216     }
```

ソースコード 17 io/github/massongit/othello2017/kotlin/app/play/information/Information.fxml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.scene.control.Label?>
4  <Label fx:id="information" text="%play-first-tern" stylesheets="@Information.css"
5         xmlns="http://javafx.com/javafx/8.0.112" xmlns:fx="http://javafx.com/fxml
               /1"
6         fx:controller="io.github.massongit.othello2017.kotlin.app.play.information
               .InformationController"/>
```

ソースコード 18 io/github/massongit/othello2017/kotlin/app/play/information/Information.css

```
1  /*情報ラベル*/
2  #information {
3      -fx-font-size: 14px;
4      -fx-padding: 5 0 5 0;
5  }
```

ソースコード 19　io.github.massongit.othello2017.kotlin.app.play.information.InformationController

```kotlin
package io.github.massongit.othello2017.kotlin.app.play.information

import javafx.fxml.FXML
import javafx.fxml.Initializable
import javafx.scene.control.Label
import java.net.URL
import java.util.*

/**
 * 情報ラベルコントローラ
 * @author Masaya SUZUKI
 */
class InformationController : Initializable {
    /**
     * 情報ラベル
     */
    @FXML
    private lateinit var information: Label

    /**
     * リソースバンドル
     */
    private lateinit var resources: ResourceBundle

    override fun initialize(location: URL, resources: ResourceBundle) {
        this.resources = resources
    }

    /**
     * 表示する情報を変更する
     * @param key リソースバンドルのkey
     */
    fun change(key: String) {
        this.information.text = this.resources.getString(key)
    }
}
```

ソースコード 20　io.github.massongit.othello2017.kotlin.app.play.stone.Stone

```kotlin
package io.github.massongit.othello2017.kotlin.app.play.stone

import javafx.scene.shape.Circle

/**
 * 石
 * @param styleClass スタイルシートのクラス
 * @author Masaya SUZUKI
 */
```

```
10 open class Stone(styleClass: String) : Circle(20.0) {
11     init {
12         this.styleClass.add(styleClass)
13     }
14
15     /**
16      * @param stoneState 石の先手・後手
17      */
18     constructor(stoneState: StoneState) : this(stoneState.styleClass)
19 }
```

ソースコード 21　io.github.massongit.othello2017.kotlin.app.play.stone.Move

```
1  package io.github.massongit.othello2017.kotlin.app.play.stone
2
3  import javafx.event.EventHandler
4  import javafx.scene.input.MouseEvent
5  import java.math.BigInteger
6
7  /**
8   * 着手
9   * @param reversePattern 石を置いた際に反転する石が立ったビットボード
10  * @author Masaya SUZUKI
11  */
12 class Move(val reversePattern: BigInteger = BigInteger.ZERO) : Stone(STYLE_CLASS)
        {
13     companion object {
14         /**
15          * 着手のスタイルクラス
16          */
17         val STYLE_CLASS: String = "move"
18     }
19
20     /**
21      * ユーザ向けの仕様に変更する
22      * @param handler クリック時のイベントハンドラー
23      */
24     fun toUser(handler: EventHandler<in MouseEvent>) {
25         this.styleClass.add("move-for-user")
26         this.onMouseClicked = handler
27     }
28 }
```

ソースコード 22　io.github.massongit.othello2017.kotlin.app.play.stone.StoneState

```
1  package io.github.massongit.othello2017.kotlin.app.play.stone
2
3  /**
4   * 石の状態
```

```
 5   * (先手・後手を表す際にも使用)
 6   * @param decideTernKey プレイヤーのターンが決定した際の文章のkey
 7   * @param styleClass 石に対応するスタイルシートのクラス
 8   * @param winKey 勝利した際の文章のkey
 9   * @author Masaya SUZUKI
10   */
11  enum class StoneState(val decideTernKey: String, val styleClass: String, val
        winKey: String) {
12      /**
13       * 先手の石
14       */
15      PLAY_FIRST("your-stone-is-black", "play-first-tern", "play-first-win") {
16          override fun inv(): StoneState = DRAW_FIRST
17      },
18
19      /**
20       * 後手の石
21       */
22      DRAW_FIRST("your-stone-is-white", "draw-first-tern", "draw-first-win") {
23          override fun inv(): StoneState = PLAY_FIRST
24      };
25
26      /**
27       * 先手・後手を反転させる
28       */
29      abstract fun inv(): StoneState
30  }
```

ソースコード 23　io.github.massongit.othello2017.kotlin.app.play.ai.AI

```
1   package io.github.massongit.othello2017.kotlin.app.play.ai
2
3   import io.github.massongit.othello2017.kotlin.app.play.stone.Move
4   import io.github.massongit.othello2017.kotlin.app.play.stone.StoneState
5   import java.math.BigInteger
6
7   /**
8    * AI
9    * @author Masaya SUZUKI
10   */
11  interface AI {
12      /**
13       * AIによる処理を実行する
14       * @param tern ターン
15       * @param stoneBitBoard 石のビットボード
16       * @param moves 着手のリスト
17       * @return AIが選択した着手
18       */
```

```
19      fun execute(tern: StoneState, stoneBitBoard: MutableMap<StoneState,
            BigInteger>, moves: List<Move>): Move
20  }
```

ソースコード 24　io.github.massongit.othello2017.kotlin.app.play.ai.AIStrength

```
1   package io.github.massongit.othello2017.kotlin.app.play.ai
2
3   /**
4    * AIの強さ
5    * @author Masaya SUZUKI
6    */
7   enum class AIStrength {
8       /**
9        * 強いAI
10       */
11      STRONG,
12
13      /**
14       * 弱いAI
15       */
16      WEAK
17  }
```

ソースコード 25　io.github.massongit.othello2017.kotlin.app.play.ai.node.AINode

```
1   package io.github.massongit.othello2017.kotlin.app.play.ai.node
2
3   import io.github.massongit.othello2017.kotlin.app.play.Node
4   import io.github.massongit.othello2017.kotlin.app.play.stone.Move
5   import io.github.massongit.othello2017.kotlin.app.play.stone.StoneState
6   import java.math.BigInteger
7
8   /**
9    * ゲーム木探索を行うためのノード
10   * @param move 着手
11   * @param depth ゲーム木における深さ (デフォルト値: 1)
12   * @param parent 親ノード (デフォルト値: null)
13   * @author Masaya SUZUKI
14   */
15  abstract class AINode<T>(override var tern: StoneState, override val bitBoard:
        MutableMap<StoneState, BigInteger>, var depth: Int, val move: Move = Move(),
        val parent: T? = null) : Node<T>() {
16      /**
17       * ターンを実行したかどうか
18       */
19      var isExecuteTern: Boolean = false
20          get() = (this.nodeList != null)
21
```

```
22      /**
23       * ノードタイプ
24       */
25      var nodeType: AINodeType = AINodeType.NORMAL
26
27      /**
28       * パスしたかどうか
29       */
30      internal var isPass = false
31
32      /**
33       * 評価値
34       * (終局を迎えた場合、評価値 = ∞)
35       */
36      open var evaluationValue: Int? = null
37
38      override val moveList: MutableList<Move> = mutableListOf(this.move)
39
40      /**
41       * ノードのリスト
42       */
43      internal open var nodeList: List<T>? = null
44
45      /**
46       * ターンを実行する
47       * @return ターンを実行することで得られた着手のリスト
48       */
49      fun executeTern(): List<T> {
50          if (this.nodeList == null) {
51              this.nodeList = super.executeTern(this.moveList[0])
52          }
53          return this.nodeList!!
54      }
55
56      override fun endGame() {
57          // 評価値を∞にする
58          this.evaluationValue = Int.MAX_VALUE
59      }
60
61      override fun nextTernPerState(state: StoneState): List<T>? {
62          if (state == this.tern) {
63              this.isPass = true
64          }
65          return super.nextTernPerState(state)
66      }
67  }
```

ソースコード 26　io.github.massongit.othello2017.kotlin.app.play.ai.node.AINodeType

```kotlin
package io.github.massongit.othello2017.kotlin.app.play.ai.node

/**
 * ゲーム木探索を行うためのノードの種類
 * @author Masaya SUZUKI
 */
enum class AINodeType {
    /**
     * 通常ノード
     */
    NORMAL,

    /**
     * 葉ノードの親ノード
     */
    LEAF_PARENT,

    /**
     * 葉ノード
     */
    LEAF,

    /**
     * 葉ノードの子ノード
     */
    LEAF_CHILD
}
```

ソースコード 27　io.github.massongit.othello2017.kotlin.app.play.ai.nega_max.NegaMaxAI

```kotlin
package io.github.massongit.othello2017.kotlin.app.play.ai.nega_max

import io.github.massongit.othello2017.kotlin.app.play.ai.AI
import io.github.massongit.othello2017.kotlin.app.play.ai.AIStrength
import io.github.massongit.othello2017.kotlin.app.play.ai.node.AINodeType
import io.github.massongit.othello2017.kotlin.app.play.stone.Move
import io.github.massongit.othello2017.kotlin.app.play.stone.StoneState
import java.math.BigInteger
import java.util.*

/**
 * NegaMax法によるAI
 * @param strength AIの強さ
 * @author Masaya SUZUKI
 */
class NegaMaxAI(private val strength: AIStrength) : AI {
    /**
     * 探索を行う最大の深さ
     */
```

```
20      private val MAX_DEPTH: Int = 10
21
22      override fun execute(tern: StoneState, stoneBitBoard: MutableMap<StoneState,
            BigInteger>, moves: List<Move>): Move {
23          // 枝刈りの対象となるノードの親ノード
24          // (枝刈りの手法: 強いAIの場合はβカット / 弱いAIの場合はαカット)
25          var cutParentNode: NegaMaxNode? = null
26
27          // ルートノード
28          val rootNode = NegaMaxNode(tern, stoneBitBoard, 0, this.strength)
29          rootNode.nodeList = moves.map { NegaMaxNode(tern, stoneBitBoard, 1, this.
                strength, it, rootNode) }
30
31          // スタック
32          val nodeStack = ArrayDeque<NegaMaxNode>(rootNode.nodeList)
33
34          // 深さ優先探索を行う
35          while (nodeStack.isNotEmpty()) {
36              // 現在見ているノード
37              val currentNode = nodeStack.peek()
38
39              // ターンを実行したかどうか
40              val isExecuteTern = currentNode.isExecuteTern
41
42              if (cutParentNode == null && !isExecuteTern) { // 現在見ているノード
                    がβカットによる枝刈りの対象になっていないとき
43                  when (currentNode.depth) {
44                      this.MAX_DEPTH - 1 -> currentNode.nodeType = AINodeType.
                            LEAF_PARENT
45                      this.MAX_DEPTH -> currentNode.nodeType = AINodeType.LEAF
46                      this.MAX_DEPTH + 1 -> currentNode.nodeType = AINodeType.
                            LEAF_CHILD
47                  }
48              }
49
50              // 現在見ているノードが以下のいずれかを満たす場合、そのノードをスタッ
                    クからpopする
51              // * βカットによる枝刈りの対象になっている
52              // * 既に子ノードを展開済み
53              // * 葉ノードの子ノード
54              if (cutParentNode != null || isExecuteTern || currentNode.nodeType ==
                     AINodeType.LEAF_CHILD) {
55                  nodeStack.pop()
56              }
57
58              if (cutParentNode == null) { // 現在見ているノードがβカットによる枝
                    刈りの対象になっていないとき
59                  if (!isExecuteTern) { // 現在見ているノードがまだ子ノードを展開し
                        ていないとき
```

```
60                      // 子ノードのリスト
61                      val childNodes = currentNode.executeTern()
62
63                      // 現在見ているノードが葉ノードの子ノードでない場合、子ノードをスタックにpushする
64                      if (currentNode.nodeType != AINodeType.LEAF_CHILD) {
65                          for (node in childNodes) {
66                              nodeStack.push(node)
67                          }
68                      }
69                  }
70
71                  if (isExecuteTern || currentNode.nodeType == AINodeType.LEAF_CHILD) {
72                      // 親ノードの評価値を更新する
73                      currentNode.parent?.evaluationValue = currentNode.evaluationValue
74
75                      // 親ノードの親ノードの評価値<現在見ているノードの評価値が成り立つ場合、枝刈りを行う
76                      if (currentNode.parent != null && currentNode.parent.parent != null && currentNode.parent.parent.evaluationValue != null && currentNode.evaluationValue != null && ((this.strength == AIStrength.STRONG && currentNode.parent.parent.evaluationValue!! < currentNode.evaluationValue!!) || (this.strength == AIStrength.WEAK && currentNode.evaluationValue!! < currentNode.parent.parent.evaluationValue!!))) {
77                          cutParentNode = currentNode.parent
78                      }
79                  }
80
81              } else if (cutParentNode == currentNode) { // 現在見ているノードが枝刈りの対象となるノードの親ノードのとき
82                  cutParentNode = null
83              }
84          }
85
86          // 評価値が最大になっている着手を返す
87          return rootNode.nodeList!!.maxBy { it.evaluationValue!! }!!.move
88      }
89  }
```

ソースコード 28　io.github.massongit.othello2017.kotlin.app.play.ai.nega_max.NegaMaxNode

```
1  package io.github.massongit.othello2017.kotlin.app.play.ai.nega_max
2
3  import io.github.massongit.othello2017.kotlin.app.play.ai.AIStrength
4  import io.github.massongit.othello2017.kotlin.app.play.ai.node.AINode
```

```
5  import io.github.massongit.othello2017.kotlin.app.play.ai.node.AINodeType
6  import io.github.massongit.othello2017.kotlin.app.play.stone.Move
7  import io.github.massongit.othello2017.kotlin.app.play.stone.StoneState
8  import java.math.BigInteger
9
10 /**
11  * NegaMax法によるゲーム木探索を行うためのノード
12  * @param tern ターン
13  * @param stoneBitBoard 石のビットボード
14  * @param depth ゲーム木における深さ (デフォルト値: 1)
15  * @param strength AIの強さ
16  * @param move 着手 (デフォルト値: 空の着手)
17  * @param parent 親ノード (デフォルト値: null)
18  * @author Masaya SUZUKI
19  */
20 open class NegaMaxNode(tern: StoneState, stoneBitBoard: MutableMap<StoneState,
        BigInteger>, depth: Int, private val strength: AIStrength, move: Move = Move
        (), parent: NegaMaxNode? = null) : AINode<NegaMaxNode>(tern, stoneBitBoard,
        depth, move, parent) {
21     /**
22      * 評価値
23      * (終局を迎えた場合、評価値 = ∞)
24      * (強いAIの場合、葉ノードの着手の数をa、葉ノードの子ノードの着手の数をb_1,
            b_2, ..., b_nとしたとき、評価値 = a + max(-b_1, -b_2, ..., -b_n))
25      * (弱いAIの場合、葉ノードの着手の数をa、葉ノードの子ノードの着手の数をb_1,
            b_2, ..., b_nとしたとき、評価値 = a + min(-b_1, -b_2, ..., -b_n))
26      */
27     override var evaluationValue: Int? = null
28         get() {
29             if (field != null) {
30                 if (!this.isPass) { // パスしていないとき
31                     if (this.nodeType == AINodeType.LEAF) { // 葉ノードのとき
32                         return -field!! - this.moveList.size
33                     } else { // 葉ノードでないとき
34                         return -field!!
35                     }
36                 } else if (this.nodeType == AINodeType.LEAF_PARENT) { // 葉ノード
                        の親ノードでパスしたとき
37                     return field!! + this.moveList.size
38                 }
39             }
40             return field
41         }
42         set(value) {
43             if (value != null && (field == null || (this.strength == AIStrength.
                    STRONG && field!! < value) || (this.strength == AIStrength.WEAK &&
                     value < field!!))) {
44                 field = value
45             }
```

```
46          }
47
48      override fun changeTern(state: StoneState): List<NegaMaxNode> {
49          if (this.isPass && this.nodeType == AINodeType.LEAF_CHILD) { // 葉ノードの子ノードでパスした場合、評価値を0にする
50              this.evaluationValue = 0
51          } else if ((this.isPass && this.nodeType == AINodeType.LEAF) || (!this.isPass && this.nodeType == AINodeType.LEAF_CHILD)) { // 葉ノードでパスしたか、葉ノードの子ノードでパスしていない場合、評価値を着手の数にする
52              this.evaluationValue = this.moveList.size
53          }
54
55          return this.moveList.map { NegaMaxNode(state, this.bitBoard, this.depth + 1, this.strength, it, this) }
56      }
57  }
```

ソースコード 29　io.github.massongit.othello2017.kotlin.utils.XMLResourceBundleControl

```
1   package io.github.massongit.othello2017.kotlin.utils
2
3   import java.util.*
4
5   /**
6    * XML形式のリソースバンドルを読み込み可能にするコントローラ
7    * (https://github.com/seraphy/JavaFXSimpleApp/blob/master/src/jp/seraphyware/utils/XMLResourceBundleControl.java を元に作成)
8    * @author seraphy, Masaya SUZUKI
9    */
10  class XMLResourceBundleControl : ResourceBundle.Control() {
11      /**
12       * 拡張子
13       */
14      private val extension = "xml"
15
16      override fun getFormats(baseName: String): List<String> = listOf(this.extension)
17
18      override fun newBundle(baseName: String, locale: Locale, format: String, loader: ClassLoader, reload: Boolean): ResourceBundle? {
19          // 拡張子が設定されているものと一致するとき
20          if (this.extension == format) {
21              // プロパティをロードしたかどうか
22              var isLoadProperties = false
23
24              // プロパティ
25              val properties = Properties()
26
27              // ロケールと結合したリソース名を求める
```

```
28              val plainBundleName = this.toBundleName(baseName, locale)
29
30              // プロパティをロードする
31              // (実行しているOSのプロパティがあればそちらを優先的にロードする)
32              for (bundleName in listOf<String>(plainBundleName, listOf<String>(
                    plainBundleName, System.getProperty("os.name").toLowerCase(Locale.
                    ENGLISH).replace(" ", "")).joinToString("_"))) {
33                  // 対応するフォーマットと結合したリソース名を求める
34                  val url = loader.getResource(this.toResourceName(bundleName,
                        format))
35
36                  // プロパティを上書きロードする
37                  if (url != null) {
38                      properties.loadFromXML(url.openStream())
39                      isLoadProperties = true
40                  }
41              }
42
43              // 何らかのプロパティが読み込めた場合、プロパティをリソースバンドルに
                    接続する
44              if (isLoadProperties) {
45                  return object : ResourceBundle() {
46                      override fun handleGetObject(key: String): Any = properties.
                            getProperty(key)
47
48                      override fun getKeys(): Enumeration<String> = Collections.
                            enumeration(properties.stringPropertyNames())
49                  }
50              }
51          }
52
53          // ロードできなかった場合、nullを返す
54          return null
55      }
56  }
```

ソースコード 30　io/github/massongit/othello2017/kotlin/utils/big_integer.kt

```
1   package io.github.massongit.othello2017.kotlin.utils
2
3   import java.math.BigInteger
4
5   /**
6    * BigIntegerのビット演算子をKotlinのビット演算子と同様に扱えるよう拡張 (中置記法
         対応)
7    * @author Masaya SUZUKI
8    */
9
10  /**
```

```
11   * 論理積をとる
12   * @param v BigIntegerとの論理積をとる値
13   * @return {@code this and v}
14   */
15  infix fun BigInteger.and(v: BigInteger): BigInteger = this.and(v)
16
17  /**
18   * 論理和をとる
19   * @param v BigIntegerとの論理和をとる値
20   * @return {@code this or v}
21   */
22  infix fun BigInteger.or(v: BigInteger): BigInteger = this.or(v)
23
24  /**
25   * 排他的論理和をとる
26   * @param v BigIntegerとの排他的論理和をとる値
27   * @return {@code this xor v}
28   */
29  infix fun BigInteger.xor(v: BigInteger): BigInteger = this.xor(v)
30
31  /**
32   * 左シフトを行う
33   * {@code n} が負の値になる場合には右シフトを行う
34   * (<tt>floor(this * 2<sup>n</sup>)</tt>を算出することと同等)
35   * @param  n 左シフトを行うビット数
36   * @return {@code this shl n}
37   * @see ushr
38   */
39  infix fun BigInteger.shl(n: Int): BigInteger = this.shiftLeft(n)
40
41  /**
42   * 右シフトを行う
43   * 符号拡張が行われ、{@code n} が負の値になる場合には左シフトを行う
44   * (<tt>floor(this / 2<sup>n</sup>)</tt>を算出することと同等)
45   * @param  n 右シフトを行うビット数
46   * @return {@code this ushr n}
47   * @see shl
48   */
49  infix fun BigInteger.ushr(n: Int): BigInteger = this.shiftRight(n)
50
51  /**
52   * 否定を行う
53   * @return {@code this.inv()}
54   */
55  fun BigInteger.inv(): BigInteger = this.not()
```