```
Let two computers play against each other (first 50 moves):
------------------------------------------------------------

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR | bN | bB | bQ | bK | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 | bP | bP | bP | bP | bP | bP | bP | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    |    |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    |    |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    |    |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    |    |    |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP | wP | wP | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR | wN | wB | wQ | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

white move: wNb1-a3
black move: a7-a6
white move: wNg1-f3
black move: f7-f6
white move: wNf3-g1
black move: bKe8-f7
white move: e2-e3
black move: e7-e6
white move: wQd1-h5
black move: g7-g6

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR | bN | bB | bQ |    | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 |    | bP | bP | bP |    | bK |    | bP | 7
  +----+----+----+----+----+----+----+----+
6 | bP |    |    |    | bP | bP | bP |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    |    |    |    |    | wQ | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    |    |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 | wN |    |    |    | wP |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP | wP |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    | wB |    | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

white move: wQh5-a5
black move: b7-b6
white move: wQa5-a4
black move: b6-b5
white move: wNa3xb5
black move: bBc8-b7
white move: a2-a3
black move: a6xb5
white move: wQa4xb5
black move: bBb7-a6
```



```
white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR | bN |    | bQ |    | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 |    |    | bP | bP |    | bK |    | bP | 7
  +----+----+----+----+----+----+----+----+
6 | bB |    |    |    | bP | bP | bP |    | 6
  +----+----+----+----+----+----+----+----+
5 |    | wQ |    |    |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    |    |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 | wP |    |    |    | wP |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 |    | wP | wP | wP |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    | wB |    | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

white move: wQb5-a5
black move: bNb8-c6
white move: wQa5-a4
black move: bBa6xf1
white move: wQa4-e4
black move: d7-d5
white move: wQe4-f3
black move: bNc6-e5
white move: wQf3-g3
black move: bBf1-b5

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR |    |    | bQ |    | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 |    |    | bP |    |    | bK |    | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    |    |    | bP | bP | bP |    | 6
  +----+----+----+----+----+----+----+----+
5 |    | bB |    | bP | bN |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    |    |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 | wP |    |    |    | wP |    | wQ |    | 3
  +----+----+----+----+----+----+----+----+
2 |    | wP | wP | wP |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    | wB |    | wK |    | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

white move: a3-a4
black move: bBb5xa4
white move: c2-c3
black move: bNe5-d3
white move: wKe1-f1
black move: bBf8-d6
white move: wQg3-g4
black move: bNg8-h6
white move: wQg4-h4
black move: bNh6-f5

white to move
```

```
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR |    |    | bQ |    |    |    | bR | 8
  +----+----+----+----+----+----+----+----+
7 |    |    | bP |    |    | bK |    | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    |    | bB | bP | bP | bP |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    | bP |    | bN |    |    | 5
  +----+----+----+----+----+----+----+----+
4 | bB |    |    |    |    |    |    | wQ | 4
  +----+----+----+----+----+----+----+----+
3 |    |    | wP | bN | wP |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 |    | wP |    | wP |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    | wB |    |    | wK | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

white move: wQh4-g4
black move: bNf5-h6
white move: wQg4-h4
black move: bNh6-f5
white move: wQh4-g4
black move: bNf5-h6
white move: wQg4-h4
black move: bNh6-f5
white move: wQh4-g4
black move: bNf5-h6

    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR |    |    | bQ |    |    |    | bR | 8
  +----+----+----+----+----+----+----+----+
7 |    |    | bP |    |    | bK |    | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    |    | bB | bP | bP | bP | bN | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    | bP |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 | bB |    |    |    |    |    | wQ |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    | wP | bN | wP |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 |    | wP |    | wP |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    | wB |    |    | wK | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h


Me against computer (first rounds):
-----------------------------------

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR | bN | bB | bQ | bK | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 | bP | bP | bP | bP | bP | bP | bP | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    |    |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    |    |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
```



```
4 |    |    |    |    |    |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    |    |    |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP | wP | wP | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR | wN | wB | wQ | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

Safe:
01. a2-a3       05. b2-b3       09. d2-d3       13. f2-f3       17.
g2-g3
02. a2-a4       06. b2-b4       10. d2-d4       14. f2-f4       18.
g2-g4
03. wNb1-a3     07. c2-c3       11. e2-e3       15. wNg1-f3     19.
h2-h3
04. wNb1-c3     08. c2-c4       12. e2-e4       16. wNg1-h3     20.
h2-h4

Please enter your move: e2-e4

white move: e2-e4
black move: d7-d5

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR | bN | bB | bQ | bK | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 | bP | bP | bP |    | bP | bP | bP | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    |    |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    | bP |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    | wP |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    |    |    |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP | wP |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR | wN | wB | wQ | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

Safe:
01. a2-a3       05. b2-b3       09. wQd1-e2     13. d2-d3       17.
e4xd5       21. wBf1-b5     25. wNg1-f3     29. g2-g4
02. a2-a4       06. b2-b4       10. wQd1-f3     14. d2-d4       18.
wBf1-e2     22. wBf1-a6     26. wNg1-h3     30. h2-h3
03. wNb1-a3     07. c2-c3       11. wQd1-g4     15. wKe1-e2     19.
wBf1-d3     23. f2-f3       27. wNg1-e2     31. h2-h4
04. wNb1-c3     08. c2-c4       12. wQd1-h5     16. e4-e5       20.
wBf1-c4     24. f2-f4       28. g2-g3

Please enter your move: Nb1-c3

white move: wNb1-c3
black move: c7-c6

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR | bN | bB | bQ | bK | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
```

```
7 | bP | bP |    |    | bP | bP | bP | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    | bP |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    | bP |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    | wP |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    | wN |    |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP | wP |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    | wB | wQ | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

Safe:
01. wRa1-b1     05. b2-b4       09. wNc3-a4     13. wQd1-g4     17.
wKe1-e2     21. wBf1-d3     25. f2-f3       29. wNg1-e2     33.
h2-h4
02. a2-a3       06. wNc3-b5     10. wNc3-e2     14. wQd1-h5     18.
e4-e5       22. wBf1-c4     26. f2-f4       30. g2-g3
03. a2-a4       07. wNc3xd5     11. wQd1-e2     15. d2-d3       19.
e4xd5       23. wBf1-b5     27. wNg1-f3     31. g2-g4
04. b2-b3       08. wNc3-b1     12. wQd1-f3     16. d2-d4       20.
wBf1-e2     24. wBf1-a6     28. wNg1-h3     32. h2-h3

Please enter your move: d2-d3

white move: d2-d3
black move: bNb8-a6

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR |    | bB | bQ | bK | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 | bP | bP |    |    | bP | bP | bP | bP | 7
  +----+----+----+----+----+----+----+----+
6 | bN |    | bP |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    | bP |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    | wP |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    | wN | wP |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP |    |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    | wB | wQ | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

Safe:
01. wRa1-b1     05. b2-b4       09. wBc1-g5     13. wNc3-b1     17.
wQd1-e2     21. d3-d4       25. e4xd5       29. wNg1-f3     33.
g2-g4
02. a2-a3       06. wBc1-d2     10. wBc1-h6     14. wNc3-a4     18.
wQd1-f3     22. wKe1-e2     26. wBf1-e2     30. wNg1-h3     34.
h2-h3
03. a2-a4       07. wBc1-e3     11. wNc3-b5     15. wNc3-e2     19.
wQd1-g4     23. wKe1-d2     27. f2-f3       31. wNg1-e2     35.
h2-h4
04. b2-b3       08. wBc1-f4     12. wNc3xd5     16. wQd1-d2     20.
wQd1-h5     24. e4-e5       28. f2-f4       32. g2-g3
```



```
Please enter your move: Bc1-f4

white move: wBc1-f4
black move: bNa6-b4

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR |    | bB | bQ | bK | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 | bP | bP |    |    | bP | bP | bP | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    | bP |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    | bP |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    | bN |    |    | wP | wB |    |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    | wN | wP |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP |    |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    |    | wQ | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

Safe:
01. wRa1-b1     05. b2-b3       09. wNc3-a4     13. wQd1-b1     17.
wQd1-h5     21. e4-e5       25. wBf4-g5     29. wBf4-c7     33.
wBf4-c1     37. wNg1-e2     41. h2-h4
02. wRa1-c1     06. wNc3-b5     10. wNc3-e2     14. wQd1-e2     18.
d3-d4       22. e4xd5       26. wBf4-h6     30. wBf4-b8     34.
wBf4-g3     38. g2-g3
03. a2-a3       07. wNc3xd5     11. wQd1-d2     15. wQd1-f3     19.
wKe1-e2     23. wBf1-e2     27. wBf4-e5     31. wBf4-e3     35.
wNg1-f3     39. g2-g4
04. a2-a4       08. wNc3-b1     12. wQd1-c1     16. wQd1-g4     20.
wKe1-d2     24. f2-f3       28. wBf4-d6     32. wBf4-d2     36.
wNg1-h3     40. h2-h3

Please enter your move: a2-a3

white move: a2-a3
black move: bNb4-a6

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR |    | bB | bQ | bK | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 | bP | bP |    |    | bP | bP | bP | bP | 7
  +----+----+----+----+----+----+----+----+
6 | bN |    | bP |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    | bP |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    | wP | wB |    |    | 4
  +----+----+----+----+----+----+----+----+
3 | wP |    | wN | wP |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 |    | wP | wP |    |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    |    | wQ | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h
```

```
Safe:
01. wRa1-a2     05. b2-b3       09. wNc3-b1     13. wQd1-d2     17.
wQd1-f3     21. wKe1-e2     25. wBf1-e2     29. wBf4-e5     33.
wBf4-e3     37. wNg1-f3     41. g2-g4
02. wRa1-b1     06. b2-b4       10. wNc3-a2     14. wQd1-c1     18.
wQd1-g4     22. wKe1-d2     26. f2-f3       30. wBf4-d6     34.
wBf4-d2     38. wNg1-h3     42. h2-h3
03. wRa1-c1     07. wNc3-b5     11. wNc3-a4     15. wQd1-b1     19.
wQd1-h5     23. e4-e5       27. wBf4-g5     31. wBf4-c7     35.
wBf4-c1     39. wNg1-e2     43. h2-h4
04. a3-a4       08. wNc3xd5     12. wNc3-e2     16. wQd1-e2     20.
d3-d4       24. e4xd5       28. wBf4-h6     32. wBf4-b8     36.
wBf4-g3     40. g2-g3

Please enter your move: Qd1-g4

white move: wQd1-g4
black move: bNa6-b4

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
8 | bR |    | bB | bQ | bK | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 | bP | bP |    |    | bP | bP | bP | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    | bP |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    | bP |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    | bN |    |    | wP | wB | wQ |    | 4
  +----+----+----+----+----+----+----+----+
3 | wP |    | wN | wP |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 |    | wP | wP |    |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 | wR |    |    |    | wK | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

Safe:
01. wRa1-a2     05. a3-a4       09. wNc3xd5     13. wNc3-a4     17.
wKe1-d1     21. e4xd5       25. wBf4-h6     29. wBf4-b8     33.
wBf4-g3     37. g2-g3       41. wQg4-g3     45. wQg4-e6     49.
wQg4-e2     53. h2-h4
02. wRa1-b1     06. a3xb4       10. wNc3-b1     14. wNc3-e2     18.
wKe1-d2     22. wBf1-e2     26. wBf4-e5     30. wBf4-e3     34.
wNg1-f3     38. wQg4-g5     42. wQg4-h4     46. wQg4-d7     50.
wQg4-d1
03. wRa1-c1     07. b2-b3       11. wNc3-d1     15. d3-d4       19.
0-0-0       23. f2-f3       27. wBf4-d6     31. wBf4-d2     35.
wNg1-h3     39. wQg4-g6     43. wQg4-h5     47. wQg4xc8     51.
wQg4-h3
04. wRa1-d1     08. wNc3-b5     12. wNc3-a2     16. wKe1-e2     20.
e4-e5       24. wBf4-g5     28. wBf4-c7     32. wBf4-c1     36.
wNg1-e2     40. wQg4xg7     44. wQg4-f5     48. wQg4-f3     52.
h2-h3

Please enter your move: 0-0-0

white move: 0-0-0
black move: bNb4-a2

white to move
    a    b    c    d    e    f    g    h
  +----+----+----+----+----+----+----+----+
```



```
8 | bR |    | bB | bQ | bK | bB | bN | bR | 8
  +----+----+----+----+----+----+----+----+
7 | bP | bP |    |    | bP | bP | bP | bP | 7
  +----+----+----+----+----+----+----+----+
6 |    |    | bP |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    | bP |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    | wP | wB | wQ |    | 4
  +----+----+----+----+----+----+----+----+
3 | wP |    | wN | wP |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 | bN | wP | wP |    |    | wP | wP | wP | 2
  +----+----+----+----+----+----+----+----+
1 |    |    | wK | wR |    | wB | wN | wR | 1
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h

Safe:
01. wKc1-b1
02. wKc1-d2
03. wNc3xa2
...
```

Listing 1: source/Figure.java

```java
package org.chpr.chess.objects;

import org.chpr.chess.IBoard;
import org.chpr.chess.utils.BoardUtils;

import java.util.ArrayList;
import java.util.List;

public class Figure {
  public static final int PAWN = 1;
  public static final int ROOK = 2;
  public static final int KNIGHT = 3;
  public static final int BISHOP = 4;
  public static final int QUEEN = 5;
  public static final int KING = 6;

  public static final String PAWN_STRING = "P";
  public static final String ROOK_STRING = "R";
  public static final String KNIGHT_STRING = "N";
  public static final String BISHOP_STRING = "B";
  public static final String QUEEN_STRING = "Q";
  public static final String KING_STRING = "K";

  public static final int WHITE_OFFSET = 0;
  public static final int BLACK_OFFSET = 10;

  public static final String WHITE_STRING = "w";
  public static final String BLACK_STRING = "b";

  public static final int WHITE = 0;
  public static final int BLACK = 1;

  public static final String[] ARR_TYPE_STRING = {PAWN_STRING, ROOK_STRING,
      KNIGHT_STRING, BISHOP_STRING, QUEEN_STRING, KING_STRING};

  public static int getType(int figureIndex) {
    return figureIndex % 10;
  }

  public static int getColor(int figureIndex) {
    return figureIndex / 10;
  }

  public static String toString(int figureIndex) {
    return toString(getColor(figureIndex), getType(figureIndex));
  }

  public static String toString(int color, int type) {
    if (type == 0) {
      return "␣␣";
    }
    String ret = "";
    if (color == WHITE)
      ret = ret.concat(WHITE_STRING);
    else if (color == BLACK)
      ret = ret.concat(BLACK_STRING);
    ret = ret.concat(ARR_TYPE_STRING[type - 1]);
    return ret;
  }

  public static short fromString(String str) {
    short ret = 0;
    if (str.startsWith(WHITE_STRING)) {
      ret += WHITE_OFFSET;
      str = str.substring(1);
    }
    else if (str.startsWith(BLACK_STRING)) {
      ret += BLACK_OFFSET;
      str = str.substring(1);
    }
    for (int i = 0; i < ARR_TYPE_STRING.length; i++)
      if (str.equals(ARR_TYPE_STRING[i]))
        return (short)(ret + i + 1);
    return -1;
  }

  static public List<Move> getValidMoves(IBoard board, int col, int row) {
    List<Move> moves = new ArrayList<>();
    short[][] figures = board.getFigures();
    short figureIndex = figures[col][row];

    if (figureIndex != 0) {
      int figureType = getType(figureIndex);
      int figureColor = getColor(figureIndex);

      if (figureType == PAWN) {
        int dir = (figureColor == WHITE ? 1 : -1);
        int targetRow = row + dir;
        if (isFree(board, col, targetRow)) {
          if (targetRow == rankRow(8, figureColor)) {
            // add promotions
            moves.add(new Move(board, figureColor, QUEEN, col, row, col,
                targetRow, true, false));
            moves.add(new Move(board, figureColor, ROOK, col, row, col,
                targetRow, true, false));
            moves.add(new Move(board, figureColor, BISHOP, col, row, col,
                targetRow, true, false));
            moves.add(new Move(board, figureColor, KNIGHT, col, row, col,
                targetRow, true, false));
          } else {
            moves.add(new Move(board, figureColor, figureType, col, row, col,
                targetRow, false, false));

            if (row == rankRow(2, figureColor) && isFree(board, col, targetRow
                + dir)) {
              moves.add(new Move(board, figureColor, figureType, col, row, col,
                  targetRow + dir, false, false));
            }
          }
        }
        // check for hit
        int[] colDirs = {-1, 1};
        for (int colDir : colDirs) {
          int targetCol = col + colDir;
          if (isValidDestination(board, figureColor, targetCol, targetRow) &&
              !isFree(board, targetCol, targetRow)) {
            if (targetRow == rankRow(8, figureColor)) {
```

```
                moves.add(new Move(board, figureColor, QUEEN, col, row,
                    targetCol, targetRow, true, true));
                moves.add(new Move(board, figureColor, ROOK, col, row, targetCol,
                    targetRow, true, true));
                moves.add(new Move(board, figureColor, BISHOP, col, row,
                    targetCol, targetRow, true, true));
                moves.add(new Move(board, figureColor, KNIGHT, col, row,
                    targetCol, targetRow, true, true));
              } else {
                moves.add(new Move(board, figureColor, figureType, col, row,
                    targetCol, targetRow, false, true));
              }
            }
          }
          // check en passant
          if (board.getHistory().size() > 0) {
            Move prevMove = board.getHistory().get(board.getHistory().size() - 1);
            if (prevMove.getType() == PAWN &&
                Math.abs(prevMove.getDestRow() - prevMove.getSourceRow()) == 2 &&
                (prevMove.getDestCol() == col - 1 || prevMove.getDestCol() == col
                    + 1) &&
                row == rankRow(5, figureColor)) {
              moves.add(new Move(board, figureColor, figureType, col, row,
                  prevMove.getDestCol(), targetRow, false, true));
            }
          }
        }

        if (figureType == ROOK || figureType == QUEEN) {
          int[][] directions = {{0, 1}, {-1, 0}, {0, -1}, {1, 0}};
          moves.addAll(searchValidMoves(board, col, row, figureType, figureColor,
              directions));
        }

        if (figureType == KNIGHT) {
          int[][] deltas = {{-1, 2}, {1, 2},
              {-1, -2}, {1, -2},
              {-2, -1}, {-2, 1},
              {2, -1}, {2, 1}};
          for (int[] delta : deltas) {
            int colDelta = delta[0];
            int rowDelta = delta[1];
            if (isValidDestination(board, figureColor, col + colDelta, row +
                rowDelta)) {
              boolean hit = !isFree(board, col + colDelta, row + rowDelta);
              moves.add(new Move(board, figureColor, figureType, col, row, col +
                  colDelta, row + rowDelta, false, hit));
            }
          }
        }

        if (figureType == BISHOP || figureType == QUEEN){
          int[][] directions = {{1, 1}, {-1, 1}, {-1, -1}, {1, -1}};
          moves.addAll(searchValidMoves(board, col, row, figureType, figureColor,
              directions));
        }

        if (figureType == KING) {
          int[][] directions = {{0, 1}, {-1, 0}, {0, -1}, {1, 0}, {1, 1}, {-1,
```



```
              1}, {-1, -1}, {1, -1}};
          for (int[] direction : directions) {
            int destCol = col + direction[0];
            int destRow = row + direction[1];
            if (isValidDestination(board, figureColor, destCol, destRow)) {
              boolean hit = !isFree(board, destCol, destRow);
              moves.add(new Move(board, figureColor, figureType, col, row,
                  destCol, destRow, false, hit));
            }
          }
          if (figureColor == WHITE && board.canWhiteCastle() && col == 4 && row
              == 0) {
            short kingsideRook = figures[7][0];
            short queensideRook = figures[0][0];
            if (board.canWhiteCastleKingside()) {
              if (isFree(board, 5, 0) && isFree(board, 6, 0) &&
                  getType(kingsideRook) == ROOK && getColor(kingsideRook) ==
                  WHITE) {
                moves.add(new Move(board, figureColor, figureType, col, row, 6,
                    0, false, false));
              }
            }
            if (board.canWhiteCastleQueenside()) {
              if (isFree(board, 2, 0) && isFree(board, 3, 0) &&
                  getType(queensideRook) == ROOK && getColor(queensideRook) ==
                  WHITE) {
                moves.add(new Move(board, figureColor, figureType, col, row, 2,
                    0, false, false));
              }
            }
          }
          if (figureColor == BLACK && board.canBlackCastle() && col == 4 && row
              == 7) {
            short kingsideRook = figures[7][7];
            short queensideRook = figures[0][7];
            if (board.canBlackCastleKingside()) {
              if (isFree(board, 5, 7) && isFree(board, 6, 7) &&
                  getType(kingsideRook) == ROOK && getColor(kingsideRook) ==
                  BLACK) {
                moves.add(new Move(board, figureColor, figureType, col, row, 6,
                    7, false, false));
              }
            }
            if (board.canBlackCastleQueenside()) {
              if (isFree(board, 2, 7) && isFree(board, 3, 7) &&
                  getType(queensideRook) == ROOK && getColor(queensideRook) ==
                  BLACK) {
                moves.add(new Move(board, figureColor, figureType, col, row, 2,
                    7, false, false));
              }
            }
          }
        }
      }
    }
    return moves;
  }

  private static List<Move> searchValidMoves(IBoard board, int col, int row,
      int figureType, int figureColor, int[][] directions) {
```

```
    List<Move> moves = new ArrayList<>();
    for (int[] direction : directions) {
      int colDir = direction[0];
      int rowDir = direction[1];

      int c = col;
      int r = row;
      boolean goOn = true;
      while (goOn) {
        c += colDir;
        r += rowDir;
        if (isValidDestination(board, figureColor, c, r)) {
          boolean hit = false;
          if (!isFree(board, c, r)) {
            hit = true;
            goOn = false;
          }
          moves.add(new Move(board, figureColor, figureType, col, row, c, r,
              false, hit));
        } else {
          goOn = false;
        }
      }
    }
    return moves;
  }

  static private boolean isValidDestination(IBoard board, int color, int col,
      int row) {
    if (col < 0 || col > 7)
      return false;
    if (row < 0 || row > 7)
      return false;
    short[][] figures = board.getFigures();
    short fig = figures[col][row];
    if (color == WHITE && fig > 0 && fig < BLACK_OFFSET)
      return false;
    if (color == BLACK && fig > BLACK_OFFSET)
      return false;
    return true;
  }

  static private boolean isFree(IBoard board, int col, int row) {
    if (col < 0 || col > 7)
      return false;
    if (row < 0 || row > 7)
      return false;
    short[][] figures = board.getFigures();
    short fig = figures[col][row];
    return fig == 0;
  }

  static public List<Move> getSafeMoves(IBoard board, List<Move> moves) {
    List<Move> safeMoves = new ArrayList<>();
    for (Move move : moves) {
      IBoard b = board.cloneIncompletely();
      b.executeMove(move);
      List<Move> nextMoves =
          b.getValidMoves(BoardUtils.FlipColor(move.getColor()));
```



```
      boolean isSafe = true;
      for (Move nextMove : nextMoves) {
        if (hitsKing(b, nextMove)) {
          isSafe = false;
          break;
        }
      }
      if (isSafe) {
        safeMoves.add(move);
      }
    }
    return safeMoves;
  }

  static public boolean hitsKing(IBoard board, Move m) {
    if (m.isHit()) {
      return getType(board.getFigures()[m.getDestCol()][m.getDestRow()]) ==
          KING;
    }
    return false;
  }

  /**
   * Return row index of rank from player (=color) perspective,
   * from https://en.wikipedia.org/wiki/Glossary_of_chess#rank:
   * > A row of the chessboard. In algebraic notation,
   * > ranks are numbered 1?8 starting from White's side of the board;
   * > however, players customarily refer to ranks from their own perspectives.
   * > For example: White's king and other pieces start on his or her first (or
       "back") rank,
   * > whereas Black calls the same rank the eighth rank; White's seventh rank
       is Black's second;
   * > and so on.
   * > If neither perspective is given, White's view is assumed.
   *
   * @param rank rank for which the row index should be returned
   * @param color player perspective
   * @return row index for rank from player perspective
   */
  static private int rankRow(int rank, int color) {
    int row = rank - 1;
    if (color == BLACK) {
      return 7 - row;
    }
    return row;
  }
}
```

Listing 2: source/RandomPlayer.java

```
package org.chpr.players.random;


import org.chpr.chess.IBoard;
import org.chpr.chess.objects.Move;
import org.chpr.players.Player;

import java.util.List;
import java.util.Random;

```

```java
public class RandomPlayer implements Player {

  @Override
  public double getFitness(IBoard board, int color) {
    // not needed for random player
    return 0;
  }

  @Override
  public Move chooseMove(IBoard board, int color, int milliSeconds, Random
      random) {
    List<Move> moves = board.getValidMoves(color);
    return moves.get(random.nextInt(moves.size()));
  }
}
```

Listing 3: source/HumanPlayer.java

```java
package org.chpr.players.human;

import org.chpr.chess.IBoard;
import org.chpr.chess.objects.Figure;
import org.chpr.chess.objects.Move;
import org.chpr.chess.utils.BoardUtils;
import org.chpr.players.Player;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Random;


public class HumanPlayer implements Player {

  private BufferedReader br;

  public HumanPlayer() {
    br = new BufferedReader(new InputStreamReader(System.in));
  }

  @Override
  public double getFitness(IBoard board, int color) {
    // not needed for human player
    return 0;
  }

  @Override
  public Move chooseMove(IBoard board, int color, int milliSeconds, Random
      random) {
    List<Move> moves = board.getValidMoves(color);
    // for the human player only allow moves that leaves the king save
    moves = Figure.getSafeMoves(board, moves);
    System.out.println("Safe:");
    System.out.println(BoardUtils.formatMovesList(moves));

    Move m = null;
    while (m == null) {
      try {
        System.out.print("Please enter your move: ");
        String s = br.readLine();
        if (BoardUtils.isIndex(s)) {
          int idx = Integer.parseInt(s) - 1;
          if (idx < 0 || idx > moves.size()) {
            System.out.println("Invalid index!");
          } else {
            m = moves.get(idx);
          }
        } else {
          m = Move.Import(s, board, color);
          if (!moves.contains(m)) {
            System.out.println("Invalid move!");
            m = null;
          }
        }
      } catch (Exception e) {
        System.out.println("Move not recognised!");
      }
    }
    System.out.println();
    return m;
  }
}
```

Listing 4: source/Player.java

```java
package org.chpr.players;

import org.chpr.chess.IBoard;
import org.chpr.chess.objects.Move;

public interface Player {

  /**
   * Evaluate position of board for given color
   *
   * @param board board to evaluate
   * @param color color for which to evaluate
   * @return fitness value of position
   */
  double getFitness(IBoard board, int color);

  /**
   * Choose a move for current position and color
   *
   * @param board current position
   * @param color color that should move
   * @param milliSeconds
   * @param random
   * @return chosen move
   */
  Move chooseMove(IBoard board, int color, int milliSeconds, java.util.Random
      random);
}
```

Listing 5: source/BoardUtils.java

```java
package org.chpr.chess.utils;

import org.chpr.chess.objects.Move;
```

```java
import java.util.List;

public class BoardUtils {

  private static final int MOVELIST_ROWS = 4;

  public static int FlipColor(int color) {
    return color == 0 ? 1 : 0;
  }

  public static String ColorToString(int color) {
    return color == 0 ? "white" : "black";
  }

  public static String ColumnName(int index) {
    return null;
  }

  public static String formatMovesList(List<Move> moves) {
    StringBuilder sb = new StringBuilder();

    int size = moves.size();
    int movelist_cols = (int)Math.ceil((float)size / MOVELIST_ROWS);
    for (int row = 0; row < MOVELIST_ROWS; row++) {
      int moveIdx = row;
      for (int col = 0; col < movelist_cols; col++) {
        if (moveIdx < size) {
          sb.append(String.format("%02d. %s\t\t", moveIdx + 1,
              moves.get(moveIdx)));
        }
        moveIdx += MOVELIST_ROWS;
      }
      sb.append("\n");
    }
    return sb.toString();
  }

  public static boolean isIndex(String s) {
    return s != null && s.matches("\\d+");
  }
}
```

Listing 6: source/MyPlayer.java

```java
package org.chpr.players.artificial;

import org.chpr.chess.IBoard;
import org.chpr.chess.objects.Figure;
import org.chpr.chess.objects.Move;
import org.chpr.players.Player;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;

import static java.lang.Thread.sleep;


public class MyPlayer implements Player {
```



```java
  private Map<Integer, Double> figureValues;
  private static final double KING_VALUE = 10000.0;

  public MyPlayer() {
    figureValues = new HashMap<>();
    figureValues.put(Figure.PAWN, 1.0);
    figureValues.put(Figure.KNIGHT, 3.3);
    figureValues.put(Figure.BISHOP, 3.3);
    figureValues.put(Figure.ROOK, 5.0);
    figureValues.put(Figure.QUEEN, 9.0);
    figureValues.put(Figure.KING, KING_VALUE);
  }

  @Override
  public double getFitness(IBoard board, int color) {
    double fitness = 0.0;
    short[][] figures = board.getFigures();
    for (int col = 0; col < figures.length; col++) {
      for (int row = 0; row < figures[0].length; row++) {
        short fig = figures[col][row];
        if (fig > 0) {
          int figureType = Figure.getType(fig);
          double value = figureValues.get(figureType);
          if (Figure.getColor(fig) == color) {
            fitness += value;
          } else {
            fitness -= value;
          }
        }
      }
    }
    return fitness;
  }

  @Override
  public Move chooseMove(IBoard board, int color, int milliSeconds, Random
      random) {
    Thinker thinker = new Thinker(this, board, color, random);
    Thread t = new Thread(thinker);
    t.start();
    try {
      sleep(milliSeconds);
    } catch (InterruptedException ignored) {
    }
    t.stop();
    return thinker.getBestMove();
  }
}
```

Listing 7: source/Thinker.java

```java
package org.chpr.players.artificial;

import org.chpr.chess.IBoard;
import org.chpr.chess.objects.Move;
import org.chpr.chess.utils.BoardUtils;
import org.chpr.players.Player;

import java.util.ArrayList;
```

```java
import java.util.List;
import java.util.Random;


public class Thinker implements Runnable {

  private Player player;
  private IBoard board;
  private int color;
  private Random random;
  private List<Move> bestMoves;
  private double bestFitness;

  private static final double REAL_LOW_VALUE = -10000.0;

  public Thinker(Player player, IBoard board, int color, Random random) {
    this.player = player;
    this.board = board;
    this.color = color;
    this.random = random;
    this.bestFitness = REAL_LOW_VALUE;
    bestMoves = new ArrayList<>();
  }

  @Override
  public void run() {
    int level = 1;
    while (true) {
      List<Move> curBestMoves = new ArrayList<>();
      double curBestFitness = REAL_LOW_VALUE;
      List<Move> moves = board.getValidMoves(color);

      for (Move m : moves) {
        IBoard b = board.cloneIncompletely();
        b.executeMove(m);
        double fitness = evaluate(b, BoardUtils.FlipColor(color), level - 1);
        if (fitness >= curBestFitness) {
          if (fitness > curBestFitness) {
            curBestMoves = new ArrayList<>();
            curBestFitness = fitness;
          }
          if (!curBestMoves.contains(m)) {
            curBestMoves.add(m);
          }
        }
      }
      bestMoves = curBestMoves;
      bestFitness = curBestFitness;
      level++;
    }
  }

  public double evaluate(IBoard b, int color, int level) {
    if (level == 0) {
      return player.getFitness(b, color) * -1;
    } else {
      double max = REAL_LOW_VALUE;
      List<Move> moves = b.getValidMoves(color);
      for (Move m : moves) {
```



```java
        IBoard tmp = b.cloneIncompletely();
        tmp.executeMove(m);
        double d = evaluate(tmp, BoardUtils.FlipColor(color), level - 1);

        if (d > max) {
          max = d;
        }
      }
      return max * -1;
    }
  }

  public Move getBestMove() {
    List<Move> bestEval0Moves = new ArrayList<>();
    double maxEval0 = REAL_LOW_VALUE;
    for (Move move : bestMoves) {
      IBoard tmp = board.cloneIncompletely();
      tmp.executeMove(move);
      double eval0 = player.getFitness(tmp, color);
      if (eval0 >= maxEval0) {
        if (eval0 > maxEval0) {
          bestEval0Moves = new ArrayList<>();
          maxEval0 = eval0;
        }
        bestEval0Moves.add(move);
      }
    }
    return bestEval0Moves.get(random.nextInt(bestEval0Moves.size()));
  }
}
```

Listing 8: source/Move.java

```java
// e2-e4 pawn moves from e2 to e4
// e2xf3 pawn moves from e2 to f3 and hits
// Ra1-a8 rook moves from a1 to a8
// a7xb8N pawn hits b8 and gets promoted to knight
// a7-a8Q pawn moves to a8 and gets promoted to queen
// 0-0 king side castle
// 0-0-0 queen side castle
//

package org.chpr.chess.objects;

import org.chpr.chess.IBoard;

import java.util.List;

public class Move {

  private int color;
  private final int type;
  private final int sourceCol;
  private final int sourceRow;
  private final int destCol;
  private final int destRow;
  private int fig;
  private IBoard board;
  private boolean hit;
  private boolean prom;
```

```
  public Move(IBoard board, int color, int type, int sourceCol, int sourceRow,
      int destCol, int destRow, boolean newType) {
    this(board, color, type, sourceCol, sourceRow, destCol, destRow, newType,
        false /* will be handled by setHit() */);
  }

  public Move(IBoard board, int color, int type, int sourceCol, int sourceRow,
      int destCol, int destRow, boolean newType, boolean hit) {
    this.color = color;
    this.type = type;
    this.sourceCol = sourceCol;
    this.sourceRow = sourceRow;
    this.destCol = destCol;
    this.destRow = destRow;
    this.board = board;
    this.fig = type + (color == Figure.WHITE ? Figure.WHITE_OFFSET :
        Figure.BLACK_OFFSET);
    this.hit = hit;
    this.prom = newType;
  }

  public IBoard getBoard() {
    return board;
  }

  public int getFigureIndex() {
    return fig;
  }

  public int getColor() {
    return color;
  }

  public int getType() {
    return type;
  }

  public int getSourceCol() {
    return sourceCol;
  }

  public int getSourceRow() {
    return sourceRow;
  }

  public int getDestCol() {
    return destCol;
  }

  public int getDestRow() {
    return destRow;
  }

  public void setColor(int color) {
    if (this.color != color) {
      if (color == Figure.WHITE)
        fig -= 10;
      else if (color == Figure.BLACK)
        fig += 10;
    }
    this.color = color;
  }

  public boolean isHit() {
    return hit;
  }

  public void setHit() {
    hit = true;
  }

  public boolean isProm() {
    return prom;
  }

  @Override
  public String toString() {
    String ret = "";
    if (Figure.getType(fig) == Figure.KING) {
      if (sourceCol == 4 && destCol == 6)
        return "0-0";
      if (sourceCol == 4 && destCol == 2)
        return "0-0-0";
    }
    if (Figure.getType(fig) != Figure.PAWN && !prom) {
      ret = ret.concat(Figure.toString(fig));
    }
    char[] col = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'};
    ret = ret.concat(Character.toString(col[sourceCol]));
    ret = ret.concat(Integer.toString(sourceRow + 1).concat(hit ? "x" : "-"));
    ret = ret.concat(Character.toString(col[destCol]));
    ret = ret.concat(Integer.toString(destRow + 1));
    if (prom)
      ret = ret.concat(Figure.toString(fig));

    return ret;
  }

  @Override
  public int hashCode() {
    int result = color;
    result = 31 * result + type;
    result = 31 * result + sourceCol;
    result = 31 * result + sourceRow;
    result = 31 * result + destCol;
    result = 31 * result + destRow;
    result = 31 * result + fig;
    result = 31 * result + (hit ? 1 : 0);
    result = 31 * result + (prom ? 1 : 0);
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (obj.getClass() != Move.class)
      return false;
    Move m = (Move)obj;
```

```
    if (color != m.getColor())
      return false;
//    if (!board.equals(m.getBoard()))
//      return false;
    if (sourceCol != m.getSourceCol())
      return false;
    if (sourceRow != m.getSourceRow())
      return false;
    if (destCol != m.getDestCol())
      return false;
    if (destRow != m.getDestRow())
      return false;
    if (fig != m.getFigureIndex())
      return false;
    if (hit != m.isHit())
      return false;
    if (prom != m.isProm())
      return false;
    // no need to check for color, its in figureIndex
    // no need to check for type, its in figureIndex (fig)
    return true;
  }

  public static Move Import(String str, IBoard board, int color) {
    if (str.equals("0-0")) {
      int row = (color == Figure.WHITE ? 0 : 7);
      return new Move(board, color, Figure.KING, 4, row, 6, row, false);
    }
    if (str.equals("0-0-0")) {
      int row = (color == Figure.WHITE ? 0 : 7);
      return new Move(board, color, Figure.KING, 4, row, 2, row, false);
    }
    char[] col = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'};
    int figureIndex = 0;
    if (color == Figure.BLACK)
      figureIndex += Figure.BLACK_OFFSET;
    // if str starts with a-h, then its a pawn
    for (char c : col) {
      if (str.startsWith(Character.toString(c))) {
        figureIndex += Figure.PAWN;
        // concat for equal position of cols, rows, ...
        str = "P".concat(str);
        break;
      }
    }
    if (str.startsWith(Figure.ROOK_STRING))
      figureIndex += Figure.ROOK;
    else if (str.startsWith(Figure.KNIGHT_STRING))
      figureIndex += Figure.KNIGHT;
    else if (str.startsWith(Figure.BISHOP_STRING))
      figureIndex += Figure.BISHOP;
    else if (str.startsWith(Figure.QUEEN_STRING))
      figureIndex += Figure.QUEEN;
    else if (str.startsWith(Figure.KING_STRING))
      figureIndex += Figure.KING;
    int sourceCol = str.charAt(1) - 'a';
    int sourceRow = Integer.parseInt(Character.toString(str.charAt(2))) - 1;
    boolean hit = Character.toString(str.charAt(3)).equals("x") ? true : false;
    int destCol = str.charAt(4) - 'a';
```



```
    int destRow = Integer.parseInt(Character.toString(str.charAt(5))) - 1;
    // if str.length() is 7, then it must be a promotion with the additional
       newTypeStr at the end
    if (str.length() == 7) {
      String newTypeStr = Character.toString(str.charAt(6));
      int newType = Figure.fromString(newTypeStr);
      Move m = new Move(board, color, newType, sourceCol, sourceRow, destCol,
          destRow, true);
      if (hit)
        m.setHit();
      return m;
    }
    Move m = new Move(board, color, Figure.getType(figureIndex), sourceCol,
        sourceRow, destCol, destRow, false);
    if (hit)
      m.setHit();
    return m;
  }

  public static boolean MovesListIncludesMove(List<Move> moves, Move move) {
    for (Move m : moves)
      if (move.equals(m))
        return true;
    return false;
  }
}
```

Listing 9: source/Game.java

```
package org.chpr.chess;

import org.chpr.chess.objects.Figure;
import org.chpr.chess.objects.Move;
import org.chpr.chess.utils.BoardUtils;
import org.chpr.players.Player;
import org.chpr.players.artificial.MyPlayer;
import org.chpr.players.human.HumanPlayer;

import java.util.Date;
import java.util.List;
import java.util.Random;

public class Game {
  public static void main(String[] args) {
    long seed = (new Date()).getTime();
    Random random = new Random(seed);

    Board board = new Board();

    Player whitePlayer = new HumanPlayer();
    Player blackPlayer = new MyPlayer();

    boolean whiteMat = false;
    boolean blackMat = false;
    boolean remis = false;

    int round = 1;
    int MAX_ROUNDS = 200;
    int MAX_TIME = 500;
```

```
    int currentColor = Figure.WHITE;

    while (!whiteMat && !blackMat && !remis && round < MAX_ROUNDS) {
      System.out.println(BoardUtils.ColorToString(currentColor) + " to move");
      System.out.println(board.toString());
      System.out.println();

      Move move;
      if (currentColor == Figure.WHITE) {
        move = whitePlayer.chooseMove(board, currentColor, MAX_TIME, random);
      } else {
        move = blackPlayer.chooseMove(board, currentColor, MAX_TIME, random);
      }
      System.out.println(BoardUtils.ColorToString(currentColor) + " move: " +
          move.toString());
      System.out.println();
      board.executeMove(move);

      currentColor = BoardUtils.FlipColor(currentColor);
      if (board.isMat(currentColor)) {
        if (currentColor == Figure.WHITE) {
          whiteMat = true;
        } else {
          blackMat = true;
        }
      } else {
        List<Move> validMoves = board.getValidMoves(currentColor);
        if (Figure.getSafeMoves(board, validMoves).size() == 0) {
          remis = true;
        }
      }
      round++;
    }

    if (whiteMat)
      System.out.println("Result: BLACK wins");
    if (blackMat)
      System.out.println("Result: WHITE wins");
    if (remis || round == MAX_ROUNDS)
      System.out.println("Result: REMIS");
  }
}
```

Listing 10: source/IBoard.java

```
package org.chpr.chess;

import org.chpr.chess.objects.Move;

import java.util.List;

/**
 * Interface for public methods of our chess board
 */
public interface IBoard {
  /**
   * Return all figures that are on the board
   *
   * @return array of figures
   */
  short[][] getFigures();

  /**
   * Set given figure at given position
   *
   * @param row the row where the figure should be positioned
   * @param column the column where the figure should be positioned
   * @param figure the figure that should be positioned
   */
  void setFigure(int row, int column, short figure);

  /**
   * Reset the board, i.e. set board to start position
   */
  void reset();

  /**
   * Copy board except with empty history
   *
   * @return incomplete copy of the board
   */
  IBoard cloneIncompletely();

  /**
   * Return list of all valid moves on the current board
   *
   * @return list of all valid moves
   */
  List<Move> getValidMoves();

  /**
   * Return list of all valid moves for a given color on the current board
   *
   * @param color color for which valid moves should be returned
   * @return list of all valid moves for given color
   */
  List<Move> getValidMoves(int color);

  /**
   * Return List of previously executed moves
   *
   * @return list of previously executed moves
   */
  List<Move> getHistory();

  /**
   * Execute given move on current position
   *
   * @param move move to execute
   */
  void executeMove(Move move);

  /**
   * Check if white can still castle
   *
   * @return true, if white can still castle
   */
  boolean canWhiteCastle();
```

```
75    /**
      * Check if white can still castle on queenside
      *
      * @return true, if white can still castle on queenside
      */
80    boolean canWhiteCastleQueenside();

      /**
      * Check if white can still castle on kingside
      *
85    * @return true, if white can still castle on kingside
      */
      boolean canWhiteCastleKingside();

      /**
90    * Check if black can still castle
      *
      * @return true, if black can still castle
      */
      boolean canBlackCastle();
95
      /**
      * Check if black can still castle on queenside
      *
      * @return true, if black can still castle on queenside
100   */
      boolean canBlackCastleQueenside();

      /**
      * Check if black can still castle on kingside
105   *
      * @return true, if black can still castle on kingside
      */
      boolean canBlackCastleKingside();

110   /**
      * Check if color is in mat
      *
      * @param color color which should be checked for
      * @return true, if color is in mat
115   */
      boolean isMat(int color);
}
```

Listing 11: source/Board.java

```
1   package org.chpr.chess;

    import org.chpr.chess.objects.Figure;
    import org.chpr.chess.objects.Move;
5
    import java.util.ArrayList;
    import java.util.LinkedList;
    import java.util.List;

10  public class Board implements IBoard {
      static private int COLS = 8;
      static private int ROWS = 8;
```



```
      private short[][] figures;
15    private boolean canWhiteCastleKingside = true;
      private boolean canWhiteCastleQueenside = true;
      private boolean canBlackCastleKingside = true;
      private boolean canBlackCastleQueenside = true;
      private List<Move> history;
20


      public Board() {
        this.reset();
      }
25
      @Override
      public short[][] getFigures() {
        return figures;
      }
30
      @Override
      public void setFigure(int row, int column, short figure) {
        figures[column][row] = figure;
      }
35
      /**
       * Set figures of board for better testing
       *
       * @param figures new position of board
40     */
      public void setFigures(short[][] figures) {
        this.figures = figures;
      }

45    @Override
      public void reset() {
        figures = new short[COLS][ROWS];
        // set white pieces
        figures[0][0] = Figure.ROOK + Figure.WHITE_OFFSET;
50      figures[1][0] = Figure.KNIGHT + Figure.WHITE_OFFSET;
        figures[2][0] = Figure.BISHOP + Figure.WHITE_OFFSET;
        figures[3][0] = Figure.QUEEN + Figure.WHITE_OFFSET;
        figures[4][0] = Figure.KING+ Figure.WHITE_OFFSET;
        figures[5][0] = Figure.BISHOP + Figure.WHITE_OFFSET;
55      figures[6][0] = Figure.KNIGHT+ Figure.WHITE_OFFSET;
        figures[7][0] = Figure.ROOK + Figure.WHITE_OFFSET;
        for (int col = 0; col < COLS; col++) {
          figures[col][1] = Figure.PAWN + Figure.WHITE_OFFSET;
        }
60      // set black pieces
        int lastRow = ROWS - 1;
        figures[0][lastRow] = Figure.ROOK + Figure.BLACK_OFFSET;
        figures[1][lastRow] = Figure.KNIGHT + Figure.BLACK_OFFSET;
        figures[2][lastRow] = Figure.BISHOP + Figure.BLACK_OFFSET;
65      figures[3][lastRow] = Figure.QUEEN + Figure.BLACK_OFFSET;
        figures[4][lastRow] = Figure.KING+ Figure.BLACK_OFFSET;
        figures[5][lastRow] = Figure.BISHOP + Figure.BLACK_OFFSET;
        figures[6][lastRow] = Figure.KNIGHT+ Figure.BLACK_OFFSET;
        figures[7][lastRow] = Figure.ROOK + Figure.BLACK_OFFSET;
70      for (int col = 0; col < COLS; col++) {
          figures[col][lastRow - 1] = Figure.PAWN + Figure.BLACK_OFFSET;
        }
```

```
    // reset other properties
    canWhiteCastleKingside = true;
    canWhiteCastleQueenside = true;
    canBlackCastleKingside = true;
    canBlackCastleQueenside = true;
    history = new LinkedList<>();
  }

  @Override
  public IBoard cloneIncompletely() {
    Board clonedBoard = new Board();
    for (int col = 0; col < COLS; col++) {
      for (int row = 0; row < ROWS; row++) {
        clonedBoard.figures[col][row] = figures[col][row];
      }
    }
    clonedBoard.canWhiteCastleKingside = canWhiteCastleKingside;
    clonedBoard.canWhiteCastleQueenside = canWhiteCastleQueenside;
    clonedBoard.canBlackCastleKingside = canBlackCastleKingside;
    clonedBoard.canBlackCastleQueenside = canBlackCastleQueenside;
    int historySize = history.size();
    if (historySize > 0) {
      clonedBoard.history.add(history.get(historySize - 1));
    }
    return clonedBoard;
  }

  @Override
  public List<Move> getValidMoves() {
    List<Move> moves = new ArrayList<>();
    for (int col = 0; col < COLS; col++) {
      for (int row = 0; row < ROWS; row++) {
        // iterate over every field
        short figure = figures[col][row];
        moves.addAll(Figure.getValidMoves(this, col, row));
      }
    }
    return moves;
  }

  @Override
  public List<Move> getValidMoves(int color) {
    List<Move> moves = new ArrayList<>();
    for (int col = 0; col < COLS; col++) {
      for (int row = 0; row < ROWS; row++) {
        // iterate over every field
        short figure = figures[col][row];
        if (Figure.getColor(figure) == color) {
          moves.addAll(Figure.getValidMoves(this, col, row));
        }
      }
    }
    return moves;
  }

  @Override
  public List<Move> getHistory() {
    return history;
  }
```



```
  @Override
  public void executeMove(Move move) {
    int srcCol = move.getSourceCol();
    int srcRow = move.getSourceRow();
    int destCol = move.getDestCol();
    int destRow = move.getDestRow();
    short srcFigure = figures[srcCol][srcRow];
    short destFigure = figures[destCol][destRow];
    boolean whiteMove = move.getColor() == Figure.WHITE;

    figures[destCol][destRow] = srcFigure;
    figures[srcCol][srcRow] = 0;

    if (move.isProm()) {
      // promotion
      short newFigure = (short)(move.getType() + (whiteMove ?
          Figure.WHITE_OFFSET : Figure.BLACK_OFFSET));
      figures[destCol][destRow] = newFigure;
    }
    if (move.getType() == Figure.PAWN && move.isHit() && destFigure == 0) {
      // En passant
      figures[destCol][destRow + (whiteMove ? -1 : 1)] = 0;
    }
    if (move.getType() == Figure.KING ) {
      if (Math.abs(destCol - srcCol) == 2) {
        // castle
        if (whiteMove) {
          if (destCol == 2) {
            figures[0][0] = 0;
            figures[3][0] = Figure.ROOK + Figure.WHITE_OFFSET;
          } else {
            figures[7][0] = 0;
            figures[5][0] = Figure.ROOK + Figure.WHITE_OFFSET;
          }
        } else {
          if (destCol == 2) {
            figures[0][7] = 0;
            figures[3][7] = Figure.ROOK + Figure.BLACK_OFFSET;
          } else {
            figures[7][7] = 0;
            figures[5][7] = Figure.ROOK + Figure.BLACK_OFFSET;
          }
        }
      }
      if (whiteMove) {
        canWhiteCastleQueenside = canWhiteCastleKingside = false;
      } else {
        canBlackCastleQueenside = canBlackCastleKingside = false;
      }
    }

    if (move.getType()== Figure.ROOK) {
      // if rook was moved set can castle options
      if (whiteMove) {
        if (srcCol == 0 && srcRow == 0) canWhiteCastleQueenside = false;
        if (srcCol == 7 && srcRow == 0) canWhiteCastleKingside = false;
      } else {
        if (srcCol == 0 && srcRow == 7) canBlackCastleQueenside = false;
```

```
        if (srcCol == 7 && srcRow == 7) canBlackCastleKingside = false;
      }
    }
    history.add(move);
  }

  @Override
  public boolean canWhiteCastle() {
    return canWhiteCastleKingside || canWhiteCastleQueenside;
  }

  @Override
  public boolean canWhiteCastleQueenside() {
    return canWhiteCastleQueenside;
  }

  @Override
  public boolean canWhiteCastleKingside() {
    return canWhiteCastleKingside;
  }

  @Override
  public boolean canBlackCastle() {
    return canBlackCastleKingside || canBlackCastleQueenside;
  }

  @Override
  public boolean canBlackCastleQueenside() {
    return canBlackCastleQueenside;
  }

  @Override
  public boolean canBlackCastleKingside() {
    return canBlackCastleKingside;
  }

  @Override
  public boolean isMat(int color) {
    // return true if no king of given color is on the board
    for (int row = 0; row < ROWS; row++) {
      for (int col = 0; col < COLS; col++) {
        short figure = figures[col][row];
        if (Figure.getColor(figure) == color && Figure.getType(figure) ==
            Figure.KING) {
          return false;
        }
      }
    }
    return true;
  }

  @Override
  public String toString() {
    String frame = "␣␣+----+----+----+----+----+----+----+----+\n";
    StringBuilder sb = new StringBuilder();
    sb.append("␣␣␣␣a␣␣␣␣b␣␣␣␣c␣␣␣␣d␣␣␣␣e␣␣␣␣f␣␣␣␣g␣␣␣␣h\n");
    for (int row = ROWS - 1; row >= 0; row--) {
      sb.append(frame);
      sb.append(row + 1);sb.append("␣");
```



```
      for (int col = 0; col < COLS; col++) {
        sb.append("|␣" + Figure.toString(figures[col][row]) + "␣");
      }
      sb.append("|␣" + (row + 1) + "\n");
    }
    sb.append(frame);
    sb.append("␣␣␣␣a␣␣␣␣b␣␣␣␣c␣␣␣␣d␣␣␣␣e␣␣␣␣f␣␣␣␣g␣␣␣␣h");
    return sb.toString();
  }
}
```