SWO3	**Übung zu Softwareentwicklung mit klassischen Sprachen und Bibliotheken 3**	**WS 2018/19, Angabe 8**

☒ Gruppe 1 (J. Heinzelreiter)

☐ Gruppe 2 (M. Hava) Name: Papesh Konstantin Aufwand [h]: 12

☐ Gruppe 3 (P. Kulczycki) Übungsleiter/Tutor: _____ Punkte: _____

Beispiel	Lösungsidee (max. 100%)	Implement. (max. 100%)	Testen (max. 100%)
1 (25 P + 60 P + 15 P)	100	100	100

Beispiel 1: Schach (`src/chess/`)

Implementieren Sie in C++ unter Anwendung des objektorientierten Programmierparadigmas eine einfache Version des Brettspiels „Schach“. Beachten Sie dabei die folgenden Anforderungen und Hinweise.

Für alle Spielfiguren gibt es eine gemeinsame, abstrakte Basisklasse `chessman`, die eine Schnittstelle für mindestens die folgenden Funktionalitäten bietet:

1. sie liefert die Farbe einer Spielfigur,
2. sie liefert eine symbolische Darstellung einer Spielfigur in Form eines ASCII-Zeichens,
3. sie beantwortet die Frage, ob eine Spielfigur „essentiell“ ist (ein Verlust einer essentiellen Spielfigur bedeutet das Ende des Spiels) und
4. sie beantwortet die Frage, ob sich eine Spielfigur auf einem Schachbrett von Position $\{z_0, s_0\}$ auf Position $\{z_1, s_1\}$ bewegen kann.

Alle konkreten Spielfiguren (wie z.B. Bauer und Springer) sollen dann, insbesondere für die letzte Funktionalität, eigene Implementierungen besitzen. Erstellen Sie dazu von `chessman` abgeleitete Klassen für die typischen Schachfiguren.

1. Der König („king“ – K) kann sich in alle Richtungen um ein Feld bewegen. Das Zielfeld darf aber nicht durch eine Spielfigur der eigenen Farbe besetzt sein. Der König ist eine essentielle Spielfigur.
2. Die Dame („queen“ – Q) kann sich beliebig weit diagonal, horizontal oder vertikal auf dem Spielbett bewegen, solange keine andere Spielfigur im Weg ist.
3. Der Läufer („bishop“ – B) kann sich beliebig weit diagonal auf dem Spielbett bewegen, solange keine andere Spielfigur im Weg ist.
4. Der Turm („rook“ – R) kann sich beliebig weit horizontal oder vertikal auf dem Spielbett bewegen, solange keine andere Spielfigur im Weg ist.
5. Der Springer („knight“ – N) kann sich um jeweils zwei Felder in eine Richtung (horizontal oder vertikal) und gleichzeitig um ein Feld in die andere Richtung (vertikal oder horizontal) bewegen. Dabei kann er beliebig andere Spielfiguren überspringen.
6. Der Bauer („pawn“ – P) kann sich um ein Feld nach vor bewegen. Bei seinem ersten Zug können dies auch zwei Felder nach vor sein. Der Bauer kann nur seitlich schlagen.

Alle Spielfiguren können auf einer feindlichen Spielfigur landen, was diese aus dem Spiel befördert und zum eigenen Sieg beitragen kann. Die Spielfiguren befinden sich auf einem Spielbrett was z.B. mit einer Klasse `chessboard` implementiert werden kann. Dieses Spielfeld muss mindestens die folgenden Funktionalitäten aufweisen:

1. Es muss vorgesehen sein, ein beliebig großes Spielbrett anlegen zu können (jedoch mit einer sinnvollen Mindestgröße).
2. Es muss erkannt werden, wann ein Spiel beendet ist (nämlich dann, wenn einer der Spieler eine essentielle Figur verloren hat).

3. Für jede Position muss die darauf befindliche Figur ermittelt werden können.
4. Der aktuellen Spieler sowie die Größe des Spielfeldes müssen ebenfalls abgefragt werden können.
5. Die folgenden Fragen müssen für jedes Feld des Spielbretts beantwortet werden können:
   a. Kann es „befahren“ werden bzw. ist es frei?
   b. Kann eine Spielfigur mit einer bestimmten Farbe geschlagen werden?
   c. Kann eine Spielfigur einer bestimmten Farbe darauf landen? Ist es also frei oder kann dort eine andere Figur geschlagen werden?
   d. Kann man auf diesem Feld eine Figur momentan „in die Hand nehmen“? Also ist dort eine Figur und wenn ja, von der aktuell am Zug befindlichen Farbe?
   e. Kann man die aktuell aktive („in der Hand befindliche“) Figur an diesem Feld abstellen? Darf die Figur also von ihrem aktuellen Startpunkt aus dort hinfahren und kann auch dortbleiben?
6. Kann man von einem Feld $\{z_0, s_0\}$ auf ein Feld $\{z_1, s_1\}$ fahren?
7. Folgende Aktionen müssen außerdem möglich sein:
   a. Die Spielfigur an einer bestimmten Stelle aktivieren, also „in die Hand nehmen“.
   b. Die aktive Spielfigur auf einem bestimmten Feld wieder abstellen (sofern das erlaubt ist).
8. Der aktuelle Spielstand (das Spielbrett mit den Spielfiguren) muss, genau so wie unten gezeigt, als ASCII-Grafik auf der Konsole ausgegeben werden.

Erledigen Sie nun die folgenden Aufgabenstellungen:

(a) Implementieren Sie alle klassischen Schachfiguren und testen Sie deren Bewegungsmuster ausführlich. Gehen Sie in der Lösungsidee auch darauf ein, wieviel Speicher pro Spielfigur benötigt wird.

(b) Schreiben Sie ein Hauptprogramm, mit dem zwei Spieler interaktiv Schach spielen können. Dabei müssen alle Zugmöglichkeiten der jeweils aktiven Spielfigur angezeigt werden.

Zwei Beispiele: Im linken Beispiel ist der weiße Springer (N) in Zeile 1, Spalte b aktiv. Im rechten Beispiel ist die die weiße Dame (Q) in Zeile 1, Spalte d, die einen gegnerischen schwarzen Bauern [p] in Zeile 4, Spalte d schlagen könnte, aktiv.

```
  | a  b  c  d  e  f  g  h |
--+-----------------------+--
8 | r  n  b  q  k  b  n  r | 8
7 | p  p  p  p  p  p  p  p | 7
6 | .  *  .  *  .  *  .  * | 6
5 | *  .  *  .  *  .  *  . | 5
4 | .  *  .  *  .  *  .  * | 4
3 |[*] .  [*] .  *  .  *  . | 3
2 | P  P  P  P  P  P  P  P | 2
1 | R (N) B  Q  K  B  N  R | 1
--+-----------------------+--
  | a  b  c  d  e  f  g  h |
```

```
  | a  b  c  d  e  f  g  h |
--+-----------------------+--
8 | r  n  b  q  k  b  n  r | 8
7 | p  p  p  p  *  p  p  p | 7
6 | .  *  .  *  .  *  .  * | 6
5 | *  .  *  .  *  .  * [.]| 5
4 |[.] *  . [p] .  * [.] * | 4
3 | * [.] P [.] P [.] *  . | 3
2 | P  P [.][*][.] P  P  P | 2
1 | R  N  B (Q) K  B  N  R | 1
--+-----------------------+--
  | a  b  c  d  e  f  g  h |
```

Abbildung: Die unterschiedlichen Farben der Spielfiguren sind als Groß- bzw. Kleinbuchstaben, die unterschiedlichen Farben der Felder durch Punkt bzw. Stern dargestellt. Die aktive Spielfigur ist in runde Klammern gesetzt. Die Felder, die die aktive Spielfigur betreten kann, sind in eckige Klammern gesetzt.

(c) Schreiben Sie ein alternatives Hauptprogramm (oder eine über die Kommandozeile auswählbare Variante), das bzw. die es erlaubt, einen zufälligen, und natürlich nur aus gültigen Zügen bestehenden, Spielablauf zu generieren.

SWO31 & SWB31 Softwareentwicklung mit klassischen Sprachen und Bibliotheken – WS 2018/19
Übungsabgabe 7

Konstantin Papesh

16. Dezember 2018

7.1 Schach

7.1.1 Lösungsidee

Grundsätzlich soll ein primitives Schach implementiert werden, welches die Funktionalität, die von der Angabe gefordert ist, erfüllt. Dabei sind *besondere* Schachzüge, wie etwa Roschade ausser Acht zu lassen. Weiters soll es möglich sein, den Computer gegen sich selbst Spielen zu lassen.

main.cpp

beinhaltet den Hauptteil des Programms. Hier werden die eingegebenen Argumente überprüft, so ist es möglich das Programm in einer kompatiblen Konsole mit UTF-8 aufzurufen. Dabei werden die normalen, buchstabenbasierten Figuren ausgetauscht gegen UTF-8 Symbole. Dies geschied mithilfe des Kommandozeilenarguments *-u*.
Weiters bietet das Programm das Argument *-c*, welches die Konsole nach jedem Zug cleart und das Schachbrett neu zeichnet. Dies erweitert den Komfort für den Spieler. Diese Funktionaliät wurde jedoch nur unter Linux getestet.
Auch in der *main.cpp* zu finden ist die Begrüßung und die Spielauswahl. Zur Auswahl stehen jeweils ein Multiplayer-Modus und der Simulated-Modus.

gamemodes.cpp

enthält die oben erwähnten Spielmodi. Zum einen bietet dieser File die standartkonforme Aufstellung der Schachfiguren auf einem 8x8 Brett an, als auch die beiden Spielmodi. Diese sind jeweils so ausgeführt, dass man so lange in einem Spielmode verweilt, bis ein König geschlagen wird. Erst dann returnt die Funktion.

exceptions.h

ist ein reiner Headerfile, welcher die möglichen geworfenen Exceptions enthält. Es wurden eigene Exceptions *GameOver*, *NoChessmanException* und *ChessboardTooSmall* erstellt.

global.h

beinhaltet den Datentyp *Coord*, welcher für die Adressierung auf dem Schachbrett dient. Weiters dient *global.h* als Interface für die UTF-8 und CLEAR_CONSOLE Implementierung.

chessman.h

enthält verschiedene virtuelle Klassen, welche in den Sub-Klassen implementiert werden.

chessman.cpp

ist die Klasse des *chessman*. An sich enthält diese Klasse nicht viele Methoden, da die meisten virtuell ausgeführt sind und erst in den Sub-Klassen implementiert werden. Hier befindet sich lediglich die Funktion für die Rückgabe des richtigen Symbols für die Ausgabe der Schachfigur. Hier wird überprüft, ob das klassische Symbol oder das UTF-8 Symbol zurückgegeben werden soll.

chessgame.cpp

dient als Interface zwischen der Konsole und dem Schachbrett. Hier werden Methoden angeboten, Figuren vom Schachbrett aufzuheben, niederzusetzen oder zurückzustellen. Ausserdem wird die Ausgabe des Schachbretts über diese Klasse implementiert.
Weiters speichert diese Klasse den momentanen Spieler, ob er schon eine Figur in der Hand hält und wo diese gestanden hat. Dies macht das Interfacing mit dem Schachbrett leichter.

chessboard.cpp

ist die Implementierung des Schachbretts. Primitiv ausgeführt, mehr wie ein Datentyp. Es können mit minimalen Checks[1] Figuren verschoben und hinzugefügt werden. Ausserdem bietet die Klasse eine Methode an, die Schachfigur an einer bestimmten Koordinate zurückzuliefern.
Ausserdem enthält die Klasse eine Hilfsfunktion für das Umrechnen der Eingabe[2] auf Koordinaten.

symbols.h

ist ein reiner Header-File. Dieser enthält alle möglichen Symbole für das Schach, sowohl den klassischen Buchstaben als auch die UTF-8 Symbole.
Während die Buchstaben nur einfach ausgeführt werden[3], sind die UTF-8 Symbole doppelt enthalten, einmal in schwarzer und einmal in weißer Ausführung.

[1] Nur ob der Index innerhalb des Schachfelds liegt.
[2] char + int, z.B A3
[3] Für das andere Team werden sie einfach großgestellt.

bishop.cpp

ist der Läufer. Hier wird am Anfang geprüft, ob auf dem Feld, auf das gezogen werden soll, sich eine Figur gleicher Farbe befindet. Da Spieler keine Figur gleicher Farbe schlagen können, ist hier ein *false* zurückzugeben.
Dann wird überprüft, ob das Ziehfeld auch im *Strahl* des Läufers liegt. Ist dies der Fall wird ein primitives *Ray-Casting* durchgeführt, um zu sehen ob sich eine Spielfigur zwischen dem Startfeld und dem Endfeld befindet. Ist dies der Fall kann das Feld nicht erreicht werden, da keine Spielfiguren übersprungen werden können.
Größe: 88 Byte[4]

king.cpp

ist die wichtigste Spielfigur am Feld. Wird der eigene König geschlagen, hat der Gegner gewonnen. Der König selbst darf nur ein Feld ziehen, dies jedoch 360 Grad.
Größe: 88 Byte

knight.cpp

ist der Springer. Eine Besonderheit des Springers ist es, dass er über Figuren springen kann, es muss daher nicht gerechnet werden, ob die Route zu seinem Ziel-Feld frei ist. Lediglich darf das Feld nicht von einer Figur gleicher Farbe belegt sein.
Größe: 88 Byte

pawn.cpp

implementiert den Bauern. Eine Besonderheit des Bauern ist es, dass er nicht schlagen kann wie er zieht und nicht ziehen kann wie er schlägt. Ausserdem darf der Bauer zwei Felder am Start fahren, das restliche Spiel nur ein Feld. Ausserdem darf der Bauer nur in eine Richtung ziehen, und zwar in Richtung des gegnerischen Teams.
Größe: 88 Byte

queen.cpp

ist die Figur mit den meisten möglichen Zügen. Ihr ist es möglich wie ein Turm und ein Läufer kombiniert zu ziehen. Als Erleichterung in dieser Implementierung wird einfach ein temporärer Turm erstellt und geprüft, ob dieser auf das gewünschte Feld ziehen kann. Ist dies nicht der Fall wird ein Läufer erstellt und dassselbe geprüft. Ist keiner der beiden geprüften Fälle positiv, kann auch die Dame nicht auf dieses Feld ziehen und daher wird ein *false* zurückgegeben.
Größe: 88 Byte

rook.cpp

, auch Turm genannt, kann lediglich horizontal und vertikal ziehen. Gleich wie der Läufer kann er keine Spielfiguren überspringen.

[4]Die Größe der Spielfiguren ist bedingt durch die Inkludierung von std::string in diesen. Dies ist für die UTF-8 Darstellung notwendig.

Größe: 88 Byte

7.1.2 Implementierung

Listing 7.1: main.cpp

```
1  //
2  // Created by khp on 06.12.18.
3  //
4
5  #include <iostream>
6  #include <cstring>
7  #include "gamemodes.h"
8
9  bool UTF_8 = false;
10 bool CLEAR_CONSOLE = false;
11
12 int main(int argc, char *argv[]) {
13     // parse cmd arguments
14     for (int j = 1; j < argc; ++j) {
15         char *argument = argv[j];
16         if (std::strncmp(argument, "-u", 2) == 0) {
17             std::cout << "UTF-8 Mode enabled!" << std::endl;
18             UTF_8 = true;
19         } else if (std::strncmp(argument, "-c", 2) == 0) {
20             std::cout << "Clear Mode enabled!" << std::endl;
21             CLEAR_CONSOLE = true;
22         } else {
23             std::cout << "Usage: " << argv[0] << " [options]" << std::endl;
24             std::cout << "Options:" << std::endl;
25             std::cout << "  -u   Enable UTF-8 mode." << std::endl;
26             std::cout << "  -c   Clear console after move (Linux only)." << std::
       endl;
27             return EXIT_FAILURE;
28         }
29     }
30
31     bool exit = false;
32     while(!exit) {
33         clearCmd();
34         char tChoice;
35         std::cout << "Welcome to chess v1.0" << std::endl;
36         std::cout << "Copyright 2018 by Konstantin Papesh" << std::endl << std::endl
       ;
37         std::cout << "Please select game mode:" << std::endl;
38         std::cout << "  (M) - Multiplayer" << std::endl;
39         std::cout << "  (S) - Simulated" << std::endl;
40         std::cout << "  (E) - Exit" << std::endl;
41         std::cin >> tChoice;
42         if(std::toupper(tChoice) == 'M')
43             multiplayerGame();
44         else if(std::toupper(tChoice) == 'S')
45             simulatedGame();
46         else if(std::toupper(tChoice) == 'E'){
47             exit = true;
48         }
49     }
50     return 0;
51 }
```

Listing 7.2: gamemodes.h

```
1  //
2  // Created by khp on 16.12.18.
3  //
4
5  #ifndef CHESS_GAMEMODES_H
6  #define CHESS_GAMEMODES_H
7
8  void clearCmd();
9  void multiplayerGame();
10 void simulatedGame();
11
12 #endif //CHESS_GAMEMODES_H
```

Listing 7.3: gamemodes.cpp

```
1  //
2  // Created by khp on 16.12.18.
3  //
4  #include <random>
5  #include <iostream>
6  #include "global.h"
7  #include "chessGame.h"
8  #include "chessfigures/pawn.h"
9  #include "chessfigures/bishop.h"
10 #include "chessfigures/rook.h"
11 #include "chessfigures/queen.h"
12 #include "chessfigures/king.h"
13 #include "chessfigures/knight.h"
14 #include "exceptions.h"
15 #include "gamemodes.h"
16
17 #define AMOUNT_CLEAR_LINES 8
18
19 void clearCmd() {
20     if(CLEAR_CONSOLE)
21         printf("\033c");
22     else {
23         for (int i = 0; i < AMOUNT_CLEAR_LINES; ++i) {
24             std::cout << std::endl;
25         }
26     }
27 };
28
29 void placeDefaultSetup(chessGame* cG) {
30     for (int i = 0; i < cG->getSize(); ++i) {
31         cG->placeOnBoard(toCoord(('A' + i), 2), new pawn(chessman::Colour::WHITE));
32         cG->placeOnBoard(toCoord(('A' + i), 7), new pawn(chessman::Colour::BLACK));
33     }
34     cG->placeOnBoard(toCoord('A', 1), new rook(chessman::Colour::WHITE));
35     cG->placeOnBoard(toCoord('H', 1), new rook(chessman::Colour::WHITE));
36
37     cG->placeOnBoard(toCoord('A', 8), new rook(chessman::Colour::BLACK));
38     cG->placeOnBoard(toCoord('H', 8), new rook(chessman::Colour::BLACK));
39
40     cG->placeOnBoard(toCoord('B', 1), new knight(chessman::Colour::WHITE));
41     cG->placeOnBoard(toCoord('G', 1), new knight(chessman::Colour::WHITE));
```

```
42
43      cG->placeOnBoard(toCoord('B', 8), new knight(chessman::Colour::BLACK));
44      cG->placeOnBoard(toCoord('G', 8), new knight(chessman::Colour::BLACK));
45
46      cG->placeOnBoard(toCoord('C', 1), new bishop(chessman::Colour::WHITE));
47      cG->placeOnBoard(toCoord('F', 1), new bishop(chessman::Colour::WHITE));
48
49      cG->placeOnBoard(toCoord('C', 8), new bishop(chessman::Colour::BLACK));
50      cG->placeOnBoard(toCoord('F', 8), new bishop(chessman::Colour::BLACK));
51
52      cG->placeOnBoard(toCoord('E', 8), new king(chessman::Colour::BLACK));
53      cG->placeOnBoard(toCoord('E', 1), new king(chessman::Colour::WHITE));
54
55      cG->placeOnBoard(toCoord('D', 8), new queen(chessman::Colour::BLACK));
56      cG->placeOnBoard(toCoord('D', 1), new queen(chessman::Colour::WHITE));
57  }
58
59  void multiplayerGame() {
60      chessGame cG;
61      placeDefaultSetup(&cG);
62
63      bool gameOver = false;
64      while (!gameOver) {
65          clearCmd();
66          std::cout << "Next Player is " << (cG.getPlayer() == chessman::Colour::WHITE
        ? "White" : "Black") << std::endl;
67          std::cout << cG;
68          std::string tString;
69          std::cout << "Pickup figure" << std::endl;
70          std::cin >> tString;
71          if (cG.pickupFigure(toCoord((char) tString[0], std::stoi(tString.substr(1)))
       )) {
72              clearCmd();
73              std::cout << "Next Player is " << (cG.getPlayer() == chessman::Colour::
       WHITE ? "White" : "Black") << std::endl;
74              std::cout << cG;
75              std::cout << "Place figure!" << std::endl;
76              std::cin >> tString;
77              try {
78                  if (cG.placeFigure(toCoord((char) tString[0], std::stoi(tString.
       substr(1))))) {
79                      std::cout << "Placed figure!" << std::endl;
80                  } else {
81                      std::cout << "Can't place figure here!" << std::endl;
82                  }
83              } catch (GameOverException &e) {
84                  gameOver = true;
85              }
86          } else {
87              std::cout << "Can't pick up this figure!" << std::endl;
88          }
89          cG.dropFigure();
90      }
91  }
92
93  void simulatedGame() {
94      chessGame cG;
```

```
95      placeDefaultSetup(&cG);
96
97      bool gameOver = false;
98      while (!gameOver) {
99          clearCmd();
100         std::cout << "Next Player is " << (cG.getPlayer() == chessman::Colour::WHITE
            ? "White" : "Black") << std::endl;
101         std::cout << cG;
102         srand(time(NULL));
103         while (!cG.pickupFigure(Coord(rand()%(cG.getSize()-1), rand()%(cG.getSize()
            -1))));
104         try {
105             int i = 0;
106             while(!cG.placeFigure(Coord(rand()%(cG.getSize()-1), rand()%(cG.getSize
            ()-1))) && i++ < 20);
107         } catch (GameOverException &e) {
108             gameOver = true;
109         }
110     }
111     clearCmd();
112     std::cout << "Next Player is " << (cG.getPlayer() == chessman::Colour::WHITE ? "
        White" : "Black") << std::endl;
113     std::cout << cG;
114     std::cin; // wait for enter key
115 }
```

Listing 7.4: exceptions.h

```
1  //
2  // Created by khp on 15.12.18.
3  //
4
5  #ifndef CHESS_EXCEPTIONS_H
6  #define CHESS_EXCEPTIONS_H
7
8  #include <exception>
9  #include <iostream>
10
11 class GameOverException : public std::exception {
12     virtual const char *what() const throw() {
13         return "Game over!";
14     }
15 };
16
17 class NoChessmanException : public std::exception {
18     virtual const char *what() const throw() {
19         return "No chessman to pick up!";
20     }
21 };
22
23 class ChessboardTooSmallException : public std::exception {
24     virtual const char *what() const throw() {
25         return "Board size too small!";
26     }
27 };
28
29 #endif //CHESS_EXCEPTIONS_H
```

Listing 7.5: global.h

```
1 //
2 // Created by khp on 06.12.18.
3 //
4
5 #ifndef SWO_GLOBAL_H
6 #define SWO_GLOBAL_H
7
8 #include <utility>
9
10 extern bool UTF_8;
11 extern bool CLEAR_CONSOLE;
12 typedef std::pair<unsigned int, unsigned int> Coord;
13
14 #define ASCII_TOUPPER_OFFSET -32
15 #define DEFAULT_CHESSBOARD_SIZE 8
16 #endif //SWO_GLOBAL_H
```

Listing 7.6: chessman.h

```
1 //
2 // Created by khp on 06.12.18.
3 //
4
5 #ifndef SWO_CHESSMAN_H
6 #define SWO_CHESSMAN_H
7
8
9 #include <vector>
10 #include <string>
11 #include "global.h"
12 #include "chessboard.h"
13
14 class chessboard;
15 class chessman {
16 public:
17     enum class Colour : bool {
18         WHITE, BLACK
19     };
20
21     virtual Colour getColour() const { return _colour; };
22
23     virtual std::string getSymbol() const;
24
25     virtual bool isEssential() { return _essential; };
26
27     virtual void increaseMoveCount() { _moveCount++; };
28
29     virtual bool canMoveTo(Coord from, Coord to, const chessboard *chessboard) const
       = 0;
30
31 protected:
32     Colour _colour;
33     char _symbol;
34     std::string _symbolWhiteU8;
35     std::string _symbolBlackU8;
36     bool _essential{false};
```

```
37      unsigned int _moveCount{0};
38  };
39
40
41  #endif //SWO_CHESSMAN_H
```

Listing 7.7: chessman.cpp

```
1   //
2   // Created by khp on 06.12.18.
3   //
4
5   #include "chessman.h"
6
7   std::string chessman::getSymbol() const{
8       if (UTF_8) {
9           switch (_colour) {
10              case Colour::WHITE:
11                  return _symbolWhiteU8;
12              case Colour::BLACK:
13                  return _symbolBlackU8;
14              default:
15                  break;
16          }
17      } else {
18          switch (_colour) {
19              case Colour::BLACK:
20                  return std::string{_symbol};
21              case Colour::WHITE:
22                  return std::string{(_symbol + ASCII_TOUPPER_OFFSET)};
23          }
24      }
25      return std::string{};
26  }
```

Listing 7.8: chessGame.h

```
1   //
2   // Created by khp on 13.12.18.
3   //
4
5   #ifndef CHESS_CHESSPRINTER_H
6   #define CHESS_CHESSPRINTER_H
7
8
9   #include "chessboard.h"
10  #include "global.h"
11  #include "chessman.h"
12
13  class chessGame {
14      friend std::ostream &operator<<(std::ostream &os, const chessGame &cG);
15  public:
16      chessGame(unsigned int boardSize = DEFAULT_CHESSBOARD_SIZE);
17
18      void placeOnBoard(Coord coord, chessman *chessman);
19
20      bool pickupFigure(Coord coord);
```

```
21
22      void dropFigure();
23
24      bool placeFigure(Coord coord);
25
26      chessman::Colour getPlayer() const { return _currentPlayerColour; };
27
28      unsigned int getSize() const { return _chessboard.getSize();};
29
30      std::ostream &print(std::ostream &os) const;
31
32  private:
33      unsigned int _boardSize;
34      chessboard _chessboard;
35      chessman::Colour _currentPlayerColour{chessman::Colour::WHITE};
36      chessman *_selectedFigure{nullptr};
37      Coord _selectedFigureCoord{Coord(0, 0)};
38  };
39
40
41  #endif //CHESS_CHESSPRINTER_H
```

Listing 7.9: chessGame.cpp

```
1  //
2  // Created by khp on 13.12.18.
3  //
4
5  #include <iostream>
6  #include "exceptions.h"
7  #include "chessGame.h"
8
9  chessGame::chessGame(unsigned int boardSize) : _chessboard{boardSize} {
10     std::cout << "Created chessboard with size " << boardSize << std::endl;
11 }
12
13 std::ostream &chessGame::print(std::ostream &os) const{
14     os << "  |";
15     for (int k = 0; k < _chessboard.getSize(); ++k) {
16         os << " " << char('a' + k) << " ";
17     }
18     os << "|  " << std::endl;
19     os << "--+";
20     for (int k = 0; k < _chessboard.getSize(); ++k) {
21         os << "---";
22     }
23     os << "+--" << std::endl;
24     for (int i = _chessboard.getSize()-1; i >= 0; --i) {
25         os << i + 1 << " |";
26         for (unsigned int j = 0; j < _chessboard.getSize(); ++j) {
27             chessman *tFigure = _chessboard.getChessman(Coord(j, i));
28
29
30             bool tCanMove = false;
31             if (_selectedFigure != nullptr)
32                 tCanMove = _selectedFigure->canMoveTo(_selectedFigureCoord, Coord(j,
       i), &_chessboard);
```

```
33
34              if (tFigure == nullptr){
35                  tCanMove ? os << "[" : os << " ";
36                  (i + j) % 2 ? os << "." : os << "*";
37                  tCanMove ? os << "]" : os << " ";
38              } else if(tFigure == _selectedFigure) {
39                  os << "(" << tFigure->getSymbol() << ")";
40              } else {
41                  tCanMove ? os << "[" : os << " ";
42                  os << tFigure->getSymbol();
43                  tCanMove ? os << "]" : os << " ";
44              }
45          }
46          os << "| " << i + 1<< std::endl;
47      }
48      os << "--+";
49      for (int k = 0; k < _chessboard.getSize(); ++k) {
50          os << "---";
51      }
52      os << "+--" << std::endl;
53      os << "  |";
54      for (int k = 0; k < _chessboard.getSize(); ++k) {
55          os << " " << char('a' + k) << " ";
56      }
57      os << "|  " << std::endl;
58      return os;
59  }
60
61  void chessGame::placeOnBoard(Coord coord, chessman *chessman) {
62      _chessboard.placeChessman(coord, chessman);
63  }
64
65  bool chessGame::pickupFigure(Coord coord) {
66      if (coord.first < 0 || coord.second < 0 || coord.first > _chessboard.getSize()
        ||
67          coord.second > _chessboard.getSize())
68          return false;
69      chessman *tChessman = _chessboard.getChessman(coord);
70      if (tChessman == nullptr)
71          return false;
72      else if (tChessman->getColour() != _currentPlayerColour)
73          return false;
74      _selectedFigure = tChessman;
75      _selectedFigureCoord = coord;
76      return true;
77  }
78
79  bool chessGame::placeFigure(Coord coord) {
80      if (_selectedFigure->canMoveTo(_selectedFigureCoord, coord, &_chessboard)) {
81          _chessboard.moveChessman(_selectedFigureCoord, coord);
82          _selectedFigure->increaseMoveCount();
83          _selectedFigureCoord = Coord(0, 0);
84          _selectedFigure = nullptr;
85          _currentPlayerColour == chessman::Colour::BLACK ?
86                  _currentPlayerColour = chessman::Colour::WHITE :
87                  _currentPlayerColour = chessman::Colour::BLACK;
88          return true;
```

```
89      }
90      return false;
91  }
92
93  void chessGame::dropFigure() {
94      _selectedFigure = nullptr;
95      _selectedFigureCoord = Coord(0, 0);
96  }
97
98  std::ostream &operator<<(std::ostream &os, const chessGame &cG){
99      cG.print(os);
100     return os;
101 }
```

Listing 7.10: chessboard.h

```
1  //
2  // Created by khp on 06.12.18.
3  //
4
5  #ifndef SWO_CHESSBOARD_H
6  #define SWO_CHESSBOARD_H
7
8  #include <string>
9  #include "global.h"
10 #include "chessman.h"
11
12 class chessman;
13 class chessboard {
14 public:
15     explicit chessboard(unsigned int size = DEFAULT_CHESSBOARD_SIZE);
16
17     unsigned int getSize() const;
18
19     void placeChessman(Coord coord, chessman *chessman);
20
21     void moveChessman(Coord from, Coord to);
22
23     chessman *getChessman(Coord coord) const;
24
25
26
27 private:
28     void exceptIfOutOfBounds(Coord coord) const;
29     std::vector<std::vector<chessman *>> _chessVect;
30 };
31
32 Coord toCoord(char, unsigned int);
33
34
35 #endif //SWO_CHESSBOARD_H
```

Listing 7.11: chessboard.cpp

```
1  //
2  // Created by khp on 06.12.18.
3  //
```

```
4
5  #include <stdexcept>
6  #include "chessboard.h"
7  #include "exceptions.h"
8
9  #define MIN_SIZE 6
10 chessboard::chessboard(unsigned int size) {
11     if(size < MIN_SIZE) {
12         throw ChessboardTooSmallException();
13     }
14     for (int i = 0; i < size; ++i) {
15         std::vector<chessman *> tVect;
16         for (int j = 0; j < size; ++j) {
17             tVect.push_back(nullptr);
18         }
19         _chessVect.push_back(tVect);
20     }
21 }
22
23 chessman *chessboard::getChessman(Coord coord) const {
24     exceptIfOutOfBounds(coord);
25     if (coord.first > _chessVect.size() || coord.second > _chessVect.size())
26         throw NoChessmanException();
27     return _chessVect[coord.first][coord.second];
28 }
29
30 unsigned int chessboard::getSize() const{
31     return (unsigned int) _chessVect.size();
32 }
33
34 void chessboard::placeChessman(Coord coord, chessman *chessman) {
35     exceptIfOutOfBounds(coord);
36     _chessVect[coord.first][coord.second] = chessman;
37 }
38
39 void chessboard::moveChessman(Coord from, Coord to) {
40     exceptIfOutOfBounds(from);
41     exceptIfOutOfBounds(to);
42
43     if (this->getChessman(from) == nullptr)
44         throw NoChessmanException();
45     else if (this->getChessman(to) != nullptr && this->getChessman(to)->isEssential
       ())
46         throw GameOverException();
47
48     if (_chessVect[to.first][to.second] != nullptr)
49         delete(_chessVect[to.first][to.second]);
50     _chessVect[to.first][to.second] = _chessVect[from.first][from.second];
51     _chessVect[from.first][from.second] = nullptr;
52 }
53
54 void chessboard::exceptIfOutOfBounds(Coord coord) const {
55     if (coord.first > this->getSize() || coord.first > this->getSize())
56         throw std::range_error("Out of bounds!");
57 }
58
59 Coord toCoord(char col, unsigned int row) {
```

```
60      Coord tCoord;
61      tCoord.first = std::toupper(col) - 'A';
62      tCoord.second = row - 1;
63      return tCoord;
64  }
```

Listing 7.12: symbols.h

```
1  //
2  // Created by khp on 07.12.18.
3  //
4
5  #ifndef SWO_SYMBOLS_H
6  #define SWO_SYMBOLS_H
7
8  #define SYMBOL_BISHOP 'b'
9  #define SYMBOL_KING 'k'
10 #define SYMBOL_KNIGHT 'n'
11 #define SYMBOL_PAWN 'p'
12 #define SYMBOL_QUEEN 'q'
13 #define SYMBOL_ROOK 'r'
14
15 #define SYMBOL_BISHOP_WHITE_U8 u8"\u2657"
16 #define SYMBOL_KING_WHITE_U8 u8"\u2654"
17 #define SYMBOL_KNIGHT_WHITE_U8 u8"\u2658"
18 #define SYMBOL_PAWN_WHITE_U8 u8"\u2659"
19 #define SYMBOL_QUEEN_WHITE_U8 u8"\u2655"
20 #define SYMBOL_ROOK_WHITE_U8 u8"\u2656"
21
22 #define SYMBOL_BISHOP_BLACK_U8 u8"\u265D"
23 #define SYMBOL_KING_BLACK_U8 u8"\u265A"
24 #define SYMBOL_KNIGHT_BLACK_U8 u8"\u265E"
25 #define SYMBOL_PAWN_BLACK_U8 u8"\u265F"
26 #define SYMBOL_QUEEN_BLACK_U8 u8"\u265B"
27 #define SYMBOL_ROOK_BLACK_U8 u8"\u265C"
28
29 #endif //SWO_SYMBOLS_H
```

Listing 7.13: bishop.h

```
1  //
2  // Created by khp on 06.12.18.
3  //
4
5  #ifndef SWO_BISHOP_H
6  #define SWO_BISHOP_H
7
8
9  #include "../chessman.h"
10 #include "../chessboard.h"
11
12 class bishop : public chessman {
13 public:
14     bishop(Colour colour);
15
16     virtual bool canMoveTo(Coord from, Coord to, const chessboard *chessboard) const
        override;
```

```
17 };
18
19
20 #endif //SWO_BISHOP_H
```

Listing 7.14: bishop.cpp

```
1  //
2  // Created by khp on 06.12.18.
3  //
4
5  #include "bishop.h"
6  #include "symbols.h"
7
8  bishop::bishop(Colour colour) {
9      _colour = colour;
10     _symbol = SYMBOL_BISHOP;
11     _symbolBlackU8 = SYMBOL_BISHOP_BLACK_U8;
12     _symbolWhiteU8 = SYMBOL_BISHOP_WHITE_U8;
13 }
14
15 bool bishop::canMoveTo(Coord from, Coord to, const chessboard *chessboard) const {
16     char multiplicatorRow = 1;
17     char multiplicatorCol = 1;
18     if (chessboard->getChessman(to) != nullptr && chessboard->getChessman(to)->
       getColour() == this->getColour())
19         return false;
20     if (abs(int(to.first - from.first)) - abs(int(to.second - from.second)) != 0)
21         return false;
22     if (to.first < from.first)
23         multiplicatorRow = -1;
24     if (to.second < from.second)
25         multiplicatorCol = -1;
26     for (int i = multiplicatorCol; i < to.second - from.second; i +=
       multiplicatorCol) {
27         if (chessboard->getChessman(Coord(from.first + (i * multiplicatorRow), from.
       second + (i * multiplicatorCol))) !=
28                 nullptr)
29             return false;
30     }
31     return true;
32 }
```

Listing 7.15: king.h

```
1  //
2  // Created by khp on 06.12.18.
3  //
4
5  #ifndef SWO_KING_H
6  #define SWO_KING_H
7
8
9  #include "../chessman.h"
10 #include "../global.h"
11
12 class king : public chessman {
```

```
13 public:
14     king(Colour colour);
15 
16     bool canMoveTo(Coord from, Coord to, const chessboard *chessboard) const
       override;
17 };
18 
19 
20 #endif //SWO_KING_H
```

Listing 7.16: king.cpp

```
1  //
2  // Created by khp on 06.12.18.
3  //
4  
5  #include "king.h"
6  #include "symbols.h"
7  
8  king::king(Colour colour) {
9      _colour = colour;
10     _essential = true;
11     _symbol = SYMBOL_KING;
12     _symbolBlackU8 = SYMBOL_KING_BLACK_U8;
13     _symbolWhiteU8 = SYMBOL_KING_WHITE_U8;
14 }
15 
16 bool king::canMoveTo(Coord from, Coord to, const chessboard *chessboard) const {
17     if (chessboard->getChessman(to) != nullptr && chessboard->getChessman(to)->
       getColour() == this->getColour())
18         return false;
19     return abs(int(to.first - from.first)) < 2 && abs(int(to.second - from.second))
       < 2;
20 }
```

Listing 7.17: knight.h

```
1  //
2  // Created by khp on 06.12.18.
3  //
4  
5  #ifndef SWO_KNIGHT_H
6  #define SWO_KNIGHT_H
7  
8  
9  #include "../chessman.h"
10 #include "../global.h"
11 
12 class knight : public chessman {
13 public:
14     knight(Colour colour);
15 
16     virtual bool canMoveTo(Coord from, Coord to, const chessboard *chessboard) const
        override;
17 };
18 
19 
```

```
20 #endif //SWO_KNIGHT_H
```

Listing 7.18: knight.cpp

```
1  //
2  // Created by khp on 06.12.18.
3  //
4
5  #include "knight.h"
6  #include "symbols.h"
7
8  knight::knight(Colour colour) {
9      _colour = colour;
10     _symbol = SYMBOL_KNIGHT;
11     _symbolBlackU8 = SYMBOL_KNIGHT_BLACK_U8;
12     _symbolWhiteU8 = SYMBOL_KNIGHT_WHITE_U8;
13 }
14
15 bool knight::canMoveTo(Coord from, Coord to, const chessboard *chessboard) const {
16     if (chessboard->getChessman(to) != nullptr && chessboard->getChessman(to)->
       getColour() == this->getColour())
17         return false;
18     if (to.first == from.first + 2 || to.first == from.first - 2) {
19         return to.second == from.second + 1 || to.second == from.second - 1; //x-
       axis
20     } else if (to.second == from.second + 2 || to.second == from.second - 2) { //y-
       axis
21         return to.first == from.first + 1 || to.first == from.first - 1;
22     }
23     return false;
24 }
```

Listing 7.19: pawn.h

```
1  //
2  // Created by khp on 06.12.18.
3  //
4
5  #ifndef SWO_PAWN_H
6  #define SWO_PAWN_H
7
8
9  #include "../chessman.h"
10 #include "../chessboard.h"
11
12 class pawn : public chessman {
13 public:
14     pawn(Colour colour);
15
16     bool canMoveTo(Coord from, Coord to, const chessboard *chessboard) const
       override;
17 };
18
19
20 #endif //SWO_PAWN_H
```

Listing 7.20: pawn.cpp

```
1 //
2 // Created by khp on 06.12.18.
3 //
4
5 #include "pawn.h"
6 #include "symbols.h"
7
8 pawn::pawn(Colour colour) {
9     _colour = colour;
10    _symbol = SYMBOL_PAWN;
11    _symbolBlackU8 = SYMBOL_PAWN_BLACK_U8;
12    _symbolWhiteU8 = SYMBOL_PAWN_WHITE_U8;
13 }
14
15 bool pawn::canMoveTo(Coord from, Coord to, const chessboard *chessboard) const {
16     if (chessboard->getChessman(to) != nullptr && chessboard->getChessman(to)->
       getColour() == this->getColour())
17         return false;
18     char multiplicator = 1;
19     if (this->_colour == Colour::BLACK)
20         multiplicator = -1;
21
22     if (chessboard->getChessman(to) != nullptr) {
23         if (((to.first == from.first + 1) || (to.first == from.first - 1)) && to.
       second == from.second + multiplicator)
24             return true;
25     } else if (to.second == from.second + multiplicator && to.first == from.first)
       // normal pawn move
26         return true;
27     else if (_moveCount == 0 && to.second == from.second + 2 * multiplicator &&
28              to.first == from.first) // start pawn move = 2 fields
29         return true;
30     return false;
31 }
```

Listing 7.21: queen.h

```
1 //
2 // Created by khp on 06.12.18.
3 //
4
5 #ifndef SWO_QUEEN_H
6 #define SWO_QUEEN_H
7
8
9 #include "../chessman.h"
10 #include "../global.h"
11
12 class queen : public chessman {
13 public:
14     queen(Colour colour);
15
16     bool canMoveTo(Coord from, Coord to, const chessboard *chessboard) const
       override;
17 };
18
19
```

```
20 #endif //SWO_QUEEN_H
```

Listing 7.22: queen.cpp

```
1  //
2  // Created by khp on 06.12.18.
3  //
4
5  #include "queen.h"
6  #include "symbols.h"
7  #include "rook.h"
8  #include "bishop.h"
9
10 queen::queen(Colour colour) {
11     _colour = colour;
12     _symbol = SYMBOL_QUEEN;
13     _symbolBlackU8 = SYMBOL_QUEEN_BLACK_U8;
14     _symbolWhiteU8 = SYMBOL_QUEEN_WHITE_U8;
15 }
16
17 bool queen::canMoveTo(Coord from, Coord to, const chessboard *chessboard) const {
18     if (chessboard->getChessman(to) != nullptr && chessboard->getChessman(to)->
       getColour() == this->getColour())
19         return false;
20     // lifehacking here
21     rook tRook(this->getColour());
22     if (tRook.canMoveTo(from, to, chessboard))
23         return true;
24     else {
25         bishop tBishop(this->getColour());
26         if (tBishop.canMoveTo(from, to, chessboard))
27             return true;
28     }
29     return false;
30 }
```

Listing 7.23: rook.h

```
1  //
2  // Created by khp on 06.12.18.
3  //
4
5  #ifndef SWO_ROOK_H
6  #define SWO_ROOK_H
7
8
9  #include "../chessman.h"
10 #include "../global.h"
11
12 class rook : public chessman {
13 public:
14     rook(Colour colour);
15
16     bool canMoveTo(Coord from, Coord to, const chessboard *chessboard) const
       override;
17 };
18
```

```
19
20 #endif //SWO_ROOK_H
```

Listing 7.24: rook.cpp

```
1 //
2 // Created by khp on 06.12.18.
3 //
4
5 #include "rook.h"
6 #include "symbols.h"
7
8 rook::rook(Colour colour) {
9     _colour = colour;
10    _symbol = SYMBOL_ROOK;
11    _symbolBlackU8 = SYMBOL_ROOK_BLACK_U8;
12    _symbolWhiteU8 = SYMBOL_ROOK_WHITE_U8;
13 }
14
15 bool rook::canMoveTo(Coord from, Coord to, const chessboard *chessboard) const {
16     if (chessboard->getChessman(to) != nullptr && chessboard->getChessman(to)->
       getColour() == this->getColour())
17         return false;
18     if (to.first != from.first && to.second != from.second)
19         return false;
20     else if (to.first != from.first) {
21         char multiplicator = 1;
22         if (to.first < from.first)
23             multiplicator = -1;
24         for (int i = multiplicator; abs(i) < abs(to.first - from.first); i +=
       multiplicator) {
25             if (chessboard->getChessman(Coord(from.first + i, to.second)) !=
26                 nullptr)
27                 return false;
28         }
29     } else if (to.second != from.second) {
30         char multiplicator = 1;
31         if (to.second < from.second)
32             multiplicator = -1;
33         for (int i = multiplicator; abs(i) < abs(to.second - from.second); i +=
       multiplicator) {
34             if (chessboard->getChessman(Coord(to.first, from.second + i)) !=
35                 nullptr)
36                 return false;
37         }
38     }
39     return true;
40 }
```

7.1.3 Testen

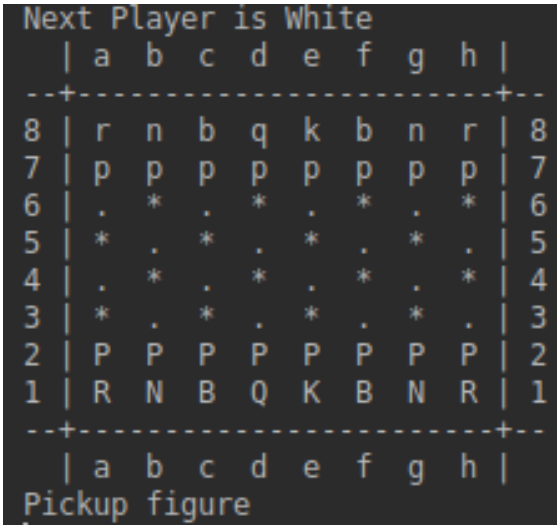

Abbildung 7.1: Schachfeld bei Beginn des Matches

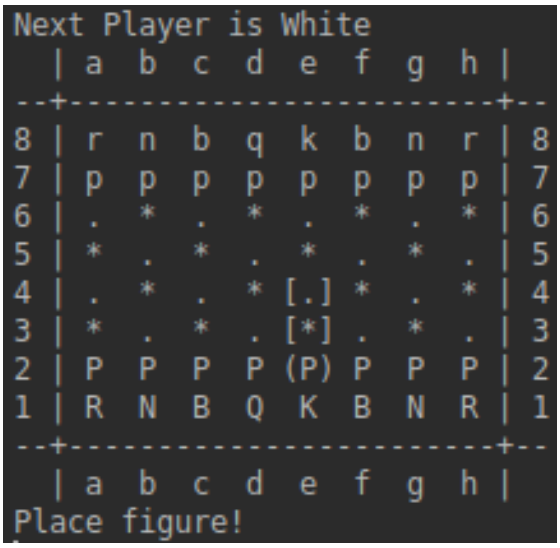

Abbildung 7.2: Auswahl eines Bauern

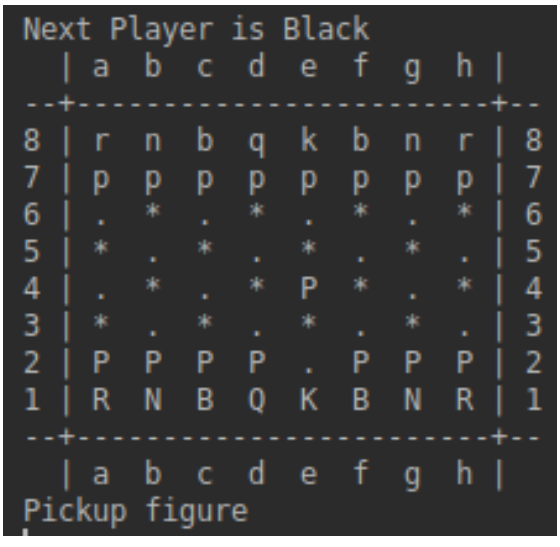

Abbildung 7.3: Zug eines Bauern

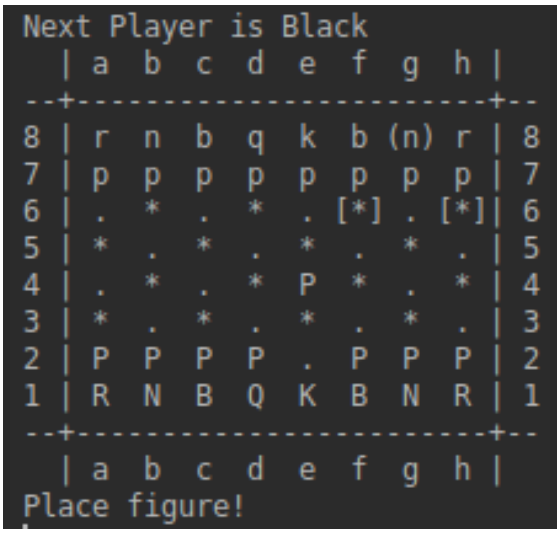

Abbildung 7.4: Gegner wählt Springer aus

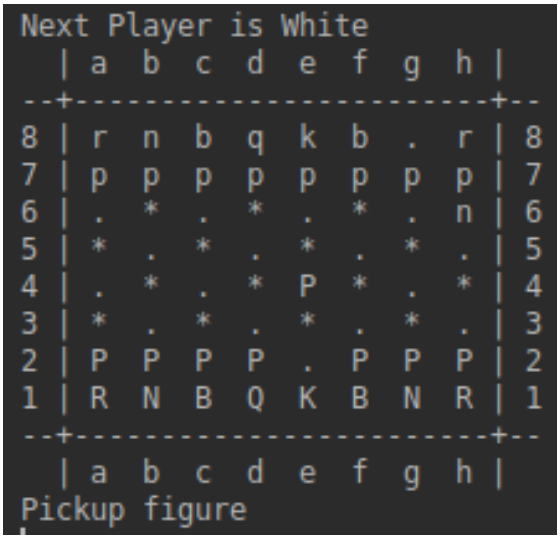

Abbildung 7.5: Zug des gegnerischen Springers

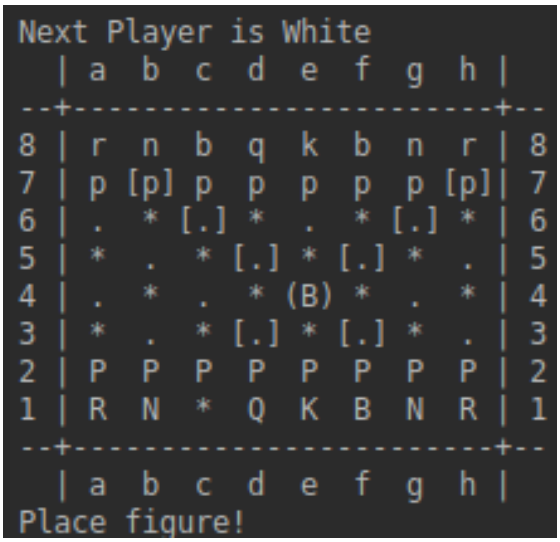

Abbildung 7.6: Mögliche Felder Läufer

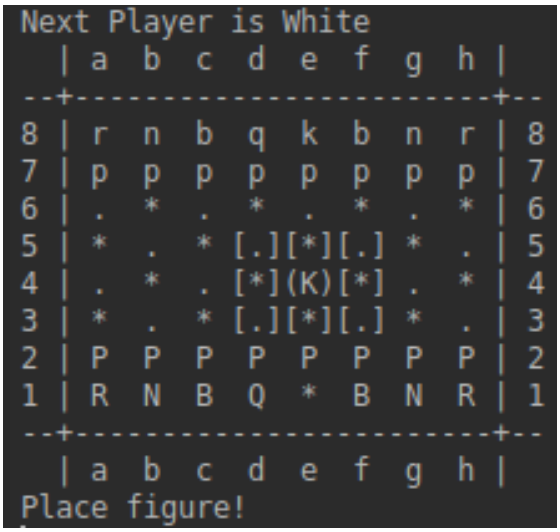

Abbildung 7.7: Mögliche Felder König

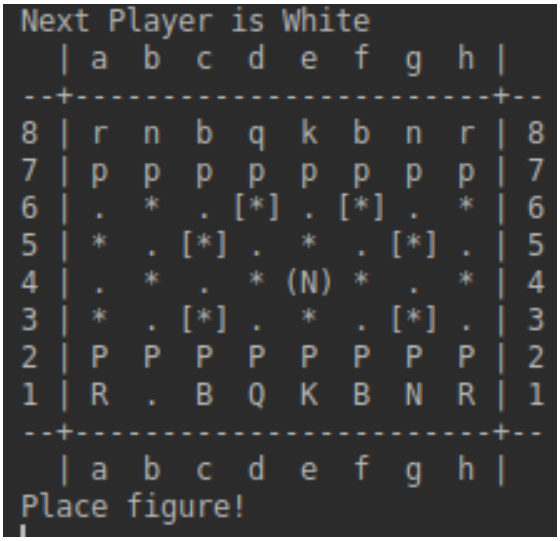

Abbildung 7.8: Mögliche Felder Springer

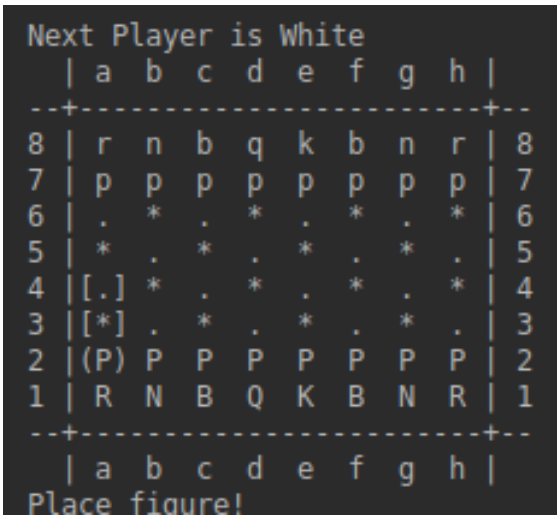

Abbildung 7.9: Mögliche Felder Bauer, erster Zug

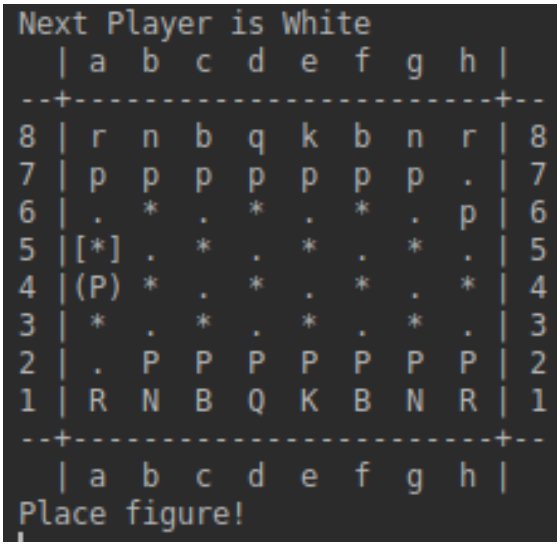

Abbildung 7.10: Mögliche Felder Bauer, zweiter Zug

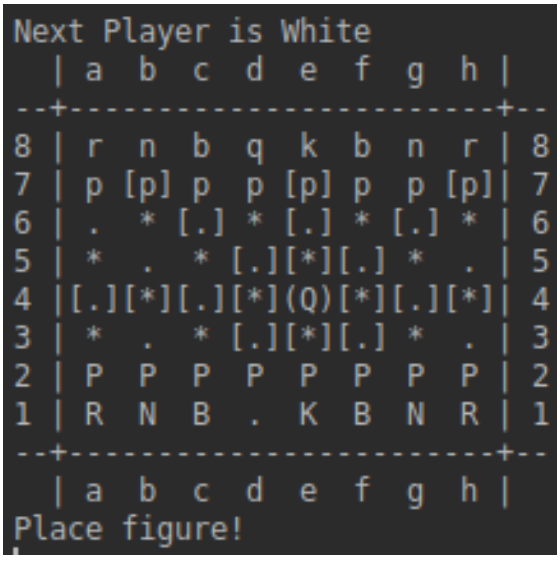

Abbildung 7.11: Mögliche Felder Königin

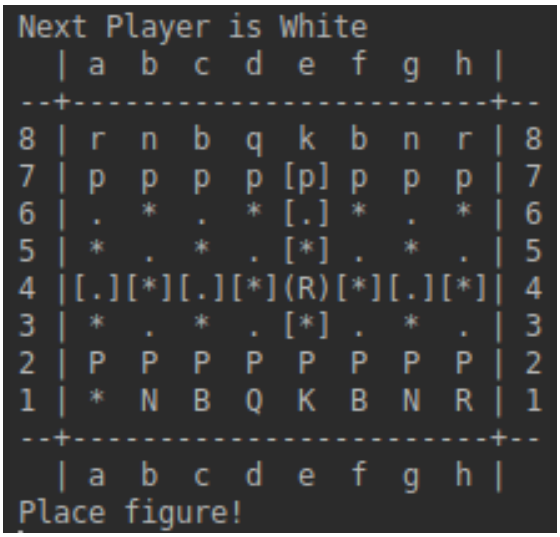

Abbildung 7.12: Mögliche Felder Turm

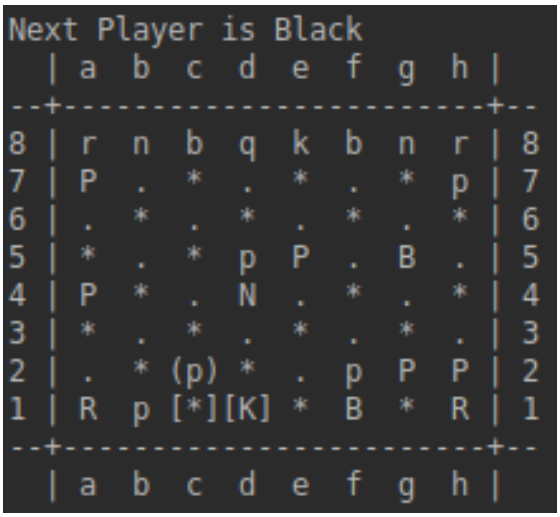

Abbildung 7.13: Ausgang eines simulierten Schachmatches

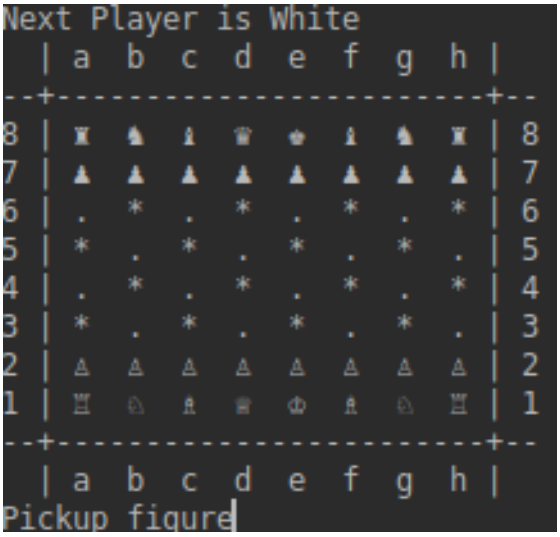

Abbildung 7.14: Darstellung des Schachbrettes in UTF-8

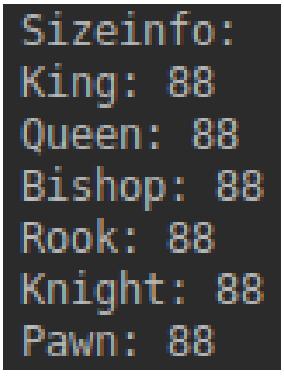

Abbildung 7.15: Größeninformation der Spielfiguren