ipso! Bildung seit 1876

Lösungsdokument

2D-Konsolenspiel [Schatzsucher]

Inhaltsverzeichnis

1 Anforderungen & Aufgabenstellung 3
1.1 Kernpunkte: 3
1.2 Erweiterte, umgesetzte Eckdaten der Lösung: 3

2 Design 3
2.1 Architekturübersicht 3
2.2 Wichtige Designentscheidungen 3

3 Implementierung 4
3.1 Projektstruktur 4
3.2 Zentrale Konstanten & Typen (Auszug) 4
3.3 Wichtige Funktionen (Kurzbeschreibung) 4
3.4 Codequalität & Robustheit 4

4 Test 5
4.1 Teststrategie 5
4.2 Testumgebung 5
4.3 Test 5
4.4 Testprotokoll 5

5 Auszüge / Screenshots 6
5.1 Ungültig - Rand 6
5.2 Ungültig – Hinderniss 7
5.3 Ungültigee Eingabe 7
5.4 Sieg 8
5.5 Beenden 8
5.6 Restart 9

6 Lessons learned 9

7 Anhang 10
7.1.1 Makefile 10
7.2 Code 11
7.2.1 main.c 11
7.2.2 Map.h 14
7.2.3 Map.c 16

1 Anforderungen & Aufgabenstellung

Ziel: Konsolenspiel (C), in dem der Spieler einen Schatz in einem Labyrinth findet.

1.1 Kernpunkte:

- 2D-Char-Array als Karte, zufällige Platzierung von Spieler (**P**), Schatz (**T**) und Hindernissen (**O**)
- Bewegung per Eingabe **W/A/S/D** (Enter zur Bestätigung)
- Nach jedem Zug wird die Karte neu gezeichnet
- Siegbedingung: Spieler erreicht Schatz → Siegesmeldung, Spielende
- Beenden (0) und Restart (1) möglich
- Saubere Modularisierung, klare Benennungen, Kommentare
- Fehlertolerante Eingaben, systematisches Testen
- Dokumentation mit: Management Summary, Anforderungen, Design, Implementierung, Test, Lessons learned, Anhang

1.2 Erweiterte, umgesetzte Eckdaten der Lösung:

- Map-Größe konfigurierbar (Standard: **15×15**)
- Hindernisdichte konfigurierbar (Standard: **12 %**)
- Getrenntes **Map-Modul** für Erzeugung, Ausgabe und Logik (Kollisionsprüfung, Update)
- Dynamische Allokation (allocateMap, freeMap)
- Robuste Einzelzeicheneingabe mit Leerraum-Unterdrückung in scanf(" %c", &input)

2 Design

2.1 Architekturübersicht

- **main.c**: Spielbegrüßung, Initialisierung, Hauptschleife, Neustart/Abbruch, runGame()
- **Map.h / Map.c**: Map-Verwaltung, Zufallsinitialisierung, Platzierung von Spieler/Schatz, Ausgaben, Bewegungs-/Kollisionslogik
- **Datenmodell**: char** map, typedef struct { int x; int y; } Position;

2.2 Wichtige Designentscheidungen

- **Modularisierung**: Trennung von Spielsteuerung und Map-Logik verbessert Testbarkeit und Wartbarkeit.
- **Dynamische Speicherverwaltung**: Flexibel für Map-Größe; sauberer Lebenszyklus (allocateMap / freeMap).
- **Eingabe**: Ein Zeichen + Enter; Leerraumpräfix in scanf vermeidet Probleme mit Newlines.
- **Determinismus bei Tests**: Für reproduzierbare Tests kann optional ein fixer Seed ergänzt werden (siehe Verbesserungen).

3 Implementierung

3.1 Projektstruktur

```
.
├── main.c          // Spiel-Loop, UI, runGame()
├── Map.c           // Map-Operationen, Kollisionslogik, Ausgabe
├── Map.h           // Schnittstellen, Typen
├── Makefile        // Build-Ziele
├── README.md       // Grundinfos/Build/Run/Steuerung
└── Dokumentation   // Lösungsdokument.pdf
```

3.2 Zentrale Konstanten & Typen (Auszug)

- **Symbole**: P (Spieler), T (Schatz), O (Hindernis), . (Freifläche)
- **Steuerung**: W/w Hoch, A/a Links, S/s Runter, D/d Rechts; 0 Abbruch, 1 Restart
- **Map**: Größe 15×15 (konfigurierbar), Hindernisdichte 12 % (konfigurierbar)
- **Datentyp**: Position { int x; int y; }

3.3 Wichtige Funktionen (Kurzbeschreibung)

- initMapModule(): Initialisiert Zufallsgenerator.
- allocateMap(h,b) / freeMap(h): Allokiert bzw. gibt Map-Speicher frei.
- createMap(...): Befüllt Map mit Hindernissen (p %) bzw. Freiflächen.
- placePlayer(...) / placeSchatz(...): Zufällige Platzierung auf gültigen Feldern, Rückgabe Position.
- printMap(...): Konsolenausgabe der Karte.
- checkMap(...): Prüf- und Update-Logik; Rückgabecode (0=Rand, 1=Hindernis, 2=Gewonnen, 3=Bewegt).

3.4 Codequalität & Robustheit

- Klare Funktionsschnittstellen in Map.h, Kommentare an allen zentralen Stellen.
- Eingaben werden validiert; ungültige Eingaben führen zu Hinweis und erneuter Abfrage.
- Speicherleckschutz: Zu jedem allocateMap existiert ein freeMap (auch bei Restart/Abbruch).

4 Test

4.1 Teststrategie

Kombination aus **funktionalen** Tests (Bedienpfade), **Grenzwerttests** (Rand/Hindernis), **Negativtests** (ungültige Eingaben) und **Nichtfunktionalem** (Speicherfreigabe, Stabilität).

4.2 Testumgebung

- OS: Windows/Linux
- Compiler: gcc (C11)
- Terminal mit UTF-8

4.3 Test

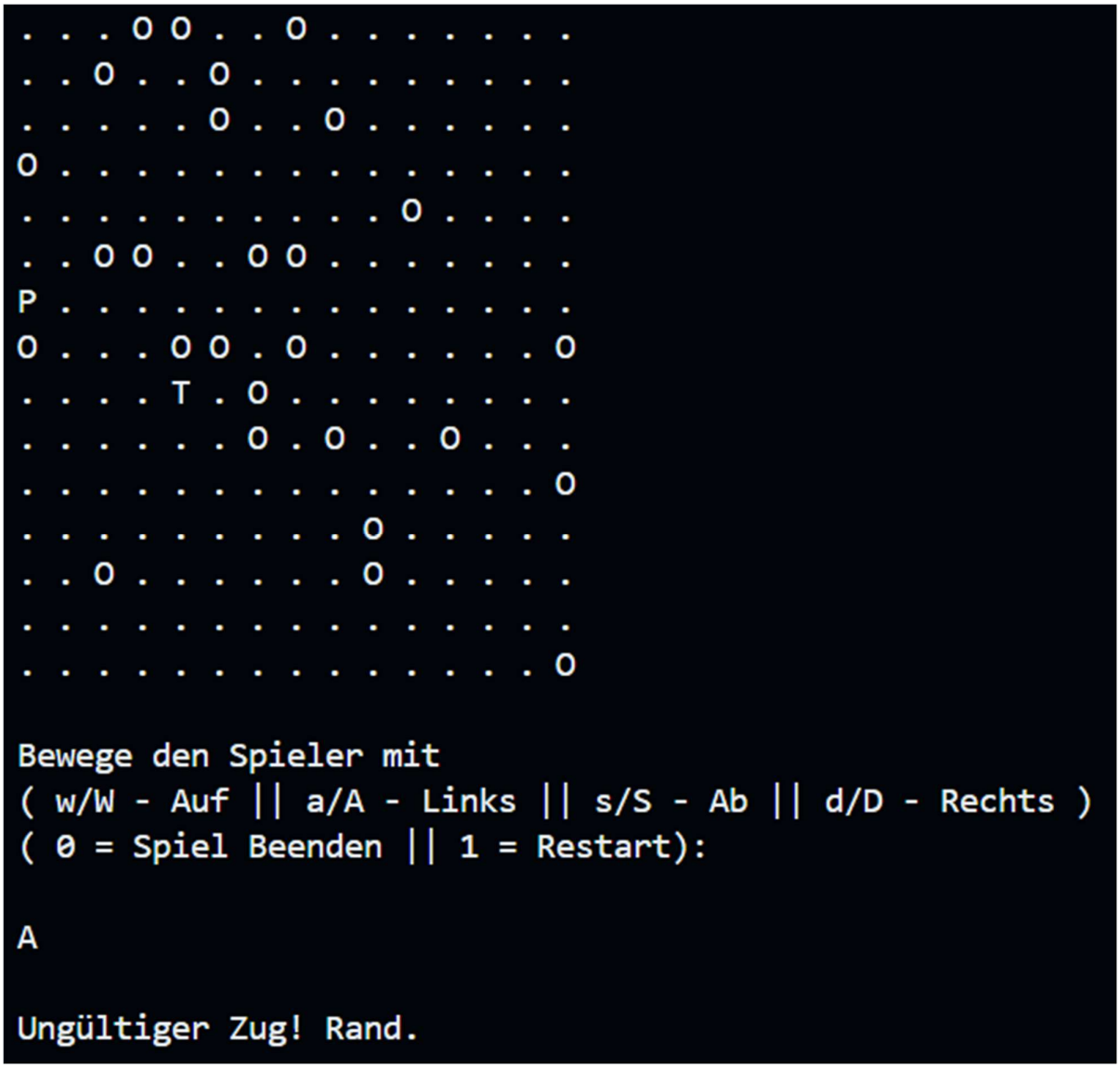
ID	Zweck	Schritte	Eingabe	Erwartetes Ergebnis
T01	Start & Anzeige	Starten	–	Begrüßung, Karte 15×15 erscheint
T02	Bewegung – frei	Auf freier Fläche bewegen	d + Enter	Spielerposition +1 in y; Karte neu gezeichnet
T03	Bewegung – Rand	Mehrfach gegen Rand laufen	a wiederholt	Hinweis „Ungültiger Zug! Rand."; Position unverändert
T04	Hindernis	In Hindernis laufen	Richtung auf O	Hinweis „Ungültiger Zug! Hindernis."; Position unverändert
T05	Schatz	Zum Schatz bewegen	Pfad zu T	Meldung „Herzlichen Glückwunsch!"; Spiel endet
T06	Ungültige Eingabe	Falsches Zeichen	X / F	Hinweis „Ungültige Eingabe!"; erneute Abfrage
T07	Restart	Neustart ausführen	1	Map-Speicher wird freigegeben; Spiel startet neu
T08	Abbruch	Beenden ausführen	0	Map-Speicher wird freigegeben; Programm endet
T09	Hindernisdichte	Sichtprüfung	–	Ca. 12 % O

4.4 Testprotokoll

Datum: 2025-10-04
T01: OK | T02: OK | T03: OK | T04: OK | T05: OK | T06: OK | T07: OK | T08: OK | T09: OK | T10: OK

5 Auszüge / Screenshots

5.1 Ungültig - Rand



5.2 Ungültig – Hinderniss

```
. . . O O . . O . . . . . . .
. . O . . O . . . . . . . . .
. . . . . O . . O . . . . . .
O . . . . . . . . . . . . . .
. . . . . . . . . . O . . . .
. . O O . . O O . . . . . . .
. . . . P . . . . . . . . . .
O . . . O O . O . . . . . . O
. . . . T . O . . . . . . . .
. . . . . . O . O . . O . . .
. . . . . . . . . . . . . . O
. . . . . . . . . O . . . . .
. . O . . . . . . O . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . O

Bewege den Spieler mit
( w/W - Auf || a/A - Links || s/S - Ab || d/D - Rechts )
( 0 = Spiel Beenden || 1 = Restart):

S

Ungültiger Zug! Hindernis. < O >
```

5.3 Ungültigee Eingabe

```
Bewege den Spieler mit
( w/W - Auf || a/A - Links || s/S - Ab || d/D - Rechts )
( 0 = Spiel Beenden || 1 = Restart):

F

Ungültige Eingabe!
```

5.4 Sieg

```
Bewege den Spieler mit
( w/W - Auf || a/A - Links || s/S - Ab || d/D - Rechts )
( 0 = Spiel Beenden || 1 = Restart):

D

. . . O O . . O . . . . . . .
. . O . . O . . . . . . . . .
. . . . . O . . O . . . . . .
O . . . . . . . . . . . . . .
. . . . . . . . . . O . . . .
. . O O . . O O . . . . . . .
. . . . . . . . . . . . . . .
O . . . O O . O . . . . . . O
. . . . P . O . . . . . . . .
. . . . . O . O . . O . . . .
. . . . . . . . . . . . . . O
. . . . . . . . . O . . . . .
. . O . . . . . . O . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . O

Herzlichen Glückwunsch! Du hast den Schatz gefunden!
```

5.5 Beenden

```
Bewege den Spieler mit
( w/W - Auf || a/A - Links || s/S - Ab || d/D - Rechts )
( 0 = Spiel Beenden || 1 = Restart):

0

Spiel wird beendet.
```

5.6 Restart

```
Bewege den Spieler mit
( w/W - Auf || a/A - Links || s/S - Ab || d/D - Rechts )
( 0 = Spiel Beenden || 1 = Restart):

1

Spiel wird neu gestartet.

Spieler ist auf Position 12 - 12.
Schatz ist auf Position 4 - 8.

O . . . . . O . . . . .
. . . . . O . . . . . .
O . . . . . O . . . . . .
O . . . . . T . . . . . .
. . . . . O
. O . . O . . O . . . . . .
. . . O . . . . . .
O . . . . . O . . . . . .
. . . O . . . . . O
. . . . . .
. O . . . . . O . . .
O . . . . . P . O .
. . . . . O . . . . . .
. . . . . .
. . . . . O . . . . . .
```

6 Lessons learned

- Konsequente **Modularisierung** vereinfacht Tests und Änderungen.
- **Dynamische Speicherverwaltung** braucht klare Besitz- und Freigaberegeln (auch auf Fehlerpfaden).
- Durch ein vorangestelltes Leerzeichen in scanf(" %c", &input) lassen sich **Leerzeichen und Whitespaces** zuverlässig ignorieren.
- Eine **Konfiguration** (Größe, Dichte) erhöht Wiederverwendung und Testbarkeit.
- Frühes Erstellen von **Diagrammen** schärft Verständnis und Kommunikation.

7 Anhang

7.1.1 Makefile

```
CC      := gcc
CFLAGS  := -std=c11 -Wall -Wextra -Wpedantic -O2
TARGET  := labyrinth
SRCS    := main.c Map.c
OBJS    := $(SRCS:.c=.o)

.PHONY: all clean run

all: $(TARGET)

$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) -o $@ $^

%.o: %.c Map.h
    $(CC) $(CFLAGS) -c $<

run: all
    ./$(TARGET)

clean:
    rm -f $(OBJS) $(TARGET)
```

7.2 Code

7.2.1 main.c

```c
#include <stdio.h>                      // Standard Ein/Ausgabe
#include <stdlib.h>                     // Standard Bibliothek

#include "Map.h"                        // Map Modul

#define Spieler 'P'                     // Char für Spieler
#define Schatz 'T'                      // Char für Schatz
#define Hindernis 'O'                   // Char für Hindernis
#define Freiflaeche '.'                 // Char für freie Fläche
#define Restart '1'                     // Char für Restart
#define Abbruch '0'                     // Char für Abbruch

#define Hoch1 'w'                       // Char 1 für Hoch
#define Runter1 's'                     // Char 1 für Runter
#define Links1 'a'                      // Char 1 für Links
#define Rechts1 'd'                     // Char 1 für Rechts

#define Hoch2 'W'                       // Char 2 für Hoch
#define Runter2 'S'                     // Char 2 für Runter
#define Links2 'A'                      // Char 2 für Links
#define Rechts2 'D'                     // Char 2 für Rechts

#define WarscheinlichkeitHindernis 12   // Warscheinlichkeit eines Hindernisses in
Prozent

#define Hoehe 15                        // Höhe der Map - Positiv < 1
#define Breite 15                       // Breite der Map - Positiv < 1


int runGame();                          // Prototyp der runGame Funktion

char input;                             // Zwischenspeicher für die Eingabe


int main() {
// Startnachricht
    printf("\nWillkommen zum Schatzsuchspiel!\n");
    printf("Ziel: Finde den Schatz '%c' auf der Karte.\n", Schatz);
    printf("Steuere den Spieler '%c' durch Eingabe von '%c/%c' (Hoch), '%c/%c'
(Links), '%c/%c' (Runter), '%c/%c' (Rechts).\n", Spieler, Hoch1, Hoch2, Links1,
Links2, Runter1, Runter2, Rechts1, Rechts2);
    printf("Vermeide Hindernisse '%c'.\n", Hindernis);
    printf("Gib '%c' ein um das Spiel zu beenden oder '%c' um neu zu starten.\n",
Abbruch, Restart);
    printf("Viel Erfolg!\n");
```

```
    initMapModule();                    // Initialisiert das Map Modul
(Zufallsgenerator)
    allocateMap(Hoehe, Breite);         // Map-Speicher Reservieren bei
Programmstart


    int restart = 0;

        while (1) {                     // Spiel ausführen
            restart = runGame();        // runGame Ausführen und Return abwarten
            if (restart != 0) break;    // Restarten bei Return(1)
        }

    freeMap(Hoehe);                     // Speicher Freigeben - Bei Programmende

    return 0;                           // Alle Instanzen beenden
}




int runGame() {

// Map erstellen/befüllen
    createMap(Hoehe, Breite, Hindernis, Freiflaeche, WarscheinlichkeitHindernis);

    Position player = placePlayer(Hoehe, Breite, Freiflaeche, Spieler); // Spieler
plazieren und Position speichern
        printf("Spieler ist auf Position %d - %d.\n", player.x+1, player.y+1);

    Position schatz = placeSchatz(Hoehe, Breite, Hindernis, Schatz, Spieler); //
Schatz plazieren und Position speichern
        printf("Schatz ist auf Position %d - %d.\n", schatz.x+1, schatz.y+1);

        // Karte ausgeben
        printMap(Hoehe, Breite);

    while (1) {

        // Eingabe(n) des Spielers anfragen und Speichern
        printf( "Bewege den Spieler mit\n"
        "( %c/%c - Auf || %c/%c - Links || %c/%c - Ab || %c/%c - Rechts )\n"
        "( %c = Spiel Beenden || %c = Restart): \n\n ",
        Hoch1, Hoch2, Links1, Links2, Runter1, Runter2, Rechts1, Rechts2, Abbruch,
Restart);
        scanf(" %c", &input); // Leerzeichen vor %c ignoriert Whitespaces wie '\n'

        // Position des Spielers zwischenspeichern für Bearbeitung bewegung
```

```
        Position newPos = player;

        // Eingabe verarbeiten
        if      (input == Hoch1   || input == Hoch2  )  newPos.x--;
        else if (input == Runter1 || input == Runter2)  newPos.x++;
        else if (input == Links1  || input == Links2 )  newPos.y--;
        else if (input == Rechts1 || input == Rechts2)  newPos.y++;

        else if (input == Abbruch) {  // Abbruch Funktion
            printf("\nSpiel wird beendet.\n");
            return 1;   // Beende alle Instanzen
        }

        else if (input == Restart) {  // Restart Funktion
            printf("\nSpiel wird neu gestartet.\n\n");
            break;   // Startet die aktuelle Main Instanz neu
        }


        else {                        // letzte option - Ungültige Eingabe
            printf("\nUngültige Eingabe!\n\n");
            continue;
        }

//
        switch (checkMap(Hoehe, Breite, player.x, player.y, newPos.x, newPos.y,
Hindernis, Freiflaeche, Spieler, Schatz)) {
            case 0: // Fehler Rand
                printf("\nUngültiger Zug! Rand.\n");
                printMap(Hoehe, Breite);                    // Karte ausgeben
                continue;                                   // Weiter
            case 1: // Fehler Hinderniss
                printf("\nUngültiger Zug! Hindernis. < %c >\n", Hindernis);
                printMap(Hoehe, Breite);                    // Karte ausgeben
                continue;                                   // Weiter
            case 2: // Gewonnen
                printMap(Hoehe, Breite);                    // Karte ausgeben
                printf("Herzlichen Glückwunsch! Du hast den Schatz gefunden!\n\n");
                return 1;                                   // Beendet alle
Instanzen
            case 3: // Bewegung erfolgreich
                printMap(Hoehe, Breite);                    // Karte ausgeben
                player.x = newPos.x;                        // Update Spieler
Position
                player.y = newPos.y;                        // Update Spieler
Position
                continue;                                   // Weiter
        }
    }
```

```
    return 0;
}
```

7.2.2 Map.h

```
#ifndef MAP_H
#define MAP_H

// Struktur für die Position
typedef struct {
    int x;
    int y;
} Position;

/**
* Initialisiert das Map Modul
* Muss vor allen anderen Funktionen aufgerufen werden
*/
void initMapModule();

/**
 * Speicher allozieren für die Map
 *
 * @param Höhe      Anzahl Zeilen
 * @param Breite    Anzahl Spalten
 */
void allocateMap(int Höhe, int Breite);

/**
 * Speicher freigeben für die Map
 *
 * @param Höhe      Anzahl Zeilen
 */
void freeMap(int Höhe);

/**
 * Erstellt eine Karte mit den Dimensionen Höhe und Breite
 * Füllt das Array zu Werscheinlichkeit in % mit dem inhalt von 'Hindernis'
 * Füllt den rest mit dem inhalt von 'Freiflaeche'
 *
 * @param Höhe                  Grösse des Arrays - map[Höhe][Breite]
 * @param Breite                Anz. Zeichen im Array - map[Höhe][Breite]
 * @param Hindernis             Inhalt für Hindernisse auf der Map - m%
Wahrscheinlichkeit
 * @param Freiflaeche           Inhalt für Freiflaeche auf der Map
 * @param Werscheinlichkeit     Prozentualer Anteil an Hindernissen
 */
```

```
void createMap(int Höhe, int Breite, char Hindernis, char Freiflaeche, int
Werscheinlichkeit);

/**
 * Nimmt die Karte mit den Dimensionen Höhe und Breite
 * Generiert die Position der Spielers zufällig
 * Plaziert den Spieler wenn da kein Hindernis oder Schatz ist
 * Giebt die Position des Spielers zurück
 *
 * @param Höhe                  Grösse des Arrays - map[Höhe][Breite]
 * @param Breite                Anz. Zeichen im Array - map[Höhe][Breite]
 * @param Freiflaeche           Inhalt für Freiflaeche auf der Map
 * @param Spielers              Inhalt des Spielers auf der Map
 */
Position placePlayer(int Höhe, int Breite, char Freiflaeche, char Spielers);

/**
 * Nimmt die Karte mit den Dimensionen Höhe und Breite
 * Generiert die Position der Schatzes zufällig
 * Plaziert den Schatz wenn da kein Hindernis oder Spieler ist
 * Giebt die Position des Schatzes zurück
 *
 * @param Höhe                  Grösse des Arrays - map[Höhe][Breite]
 * @param Breite                Anz. Zeichen im Array - map[Höhe][Breite]
 * @param Hindernis             Inhalt für Hindernis auf der Map
 * @param Schatz                Inhalt des Schatz auf der Map
 * @param Spielers              Inhalt des Spielers auf der Map
 *
 */
Position placeSchatz(int Höhe, int Breite, char Hindernis, char Schatz, char
Spielers);

/**
 * Gibt ein Char Array mit den Dimensionen Höhe * Breite auf der Konsole aus
 * Höhe = Zeilen
 * Breite = Spalten
 *
 * @param Höhe                  Grösse des Arrays - map[Höhe][Breite]
 * @param Breite                Anz. Zeichen im Array - map[Höhe][Breite]
 */
void printMap(int Höhe, int Breite);

/**
 * Updatet die Map - ohne Ausgabe
 *
 * @param Spieler_alt_Höhe       Spielerposition vor der Bewegung
 * @param Spieler_alt_Breite     Spielerposition vor der Bewegung
 * @param Spieler_neu_Höhe       Spielerposition nach der Bewegung
 * @param Spieler_neu_Breite     Spielerposition nach der Bewegung
```

```
 *
 * @param Freifläche            Inhalt für die alte Position (Freifläche)
 * @param Spieler               Inhalt für die neue Position (Spieler)
 */
void updateMap(int Spieler_alt_Höhe, int Spieler_alt_Breite, int Spieler_neu_Höhe,
int Spieler_neu_Breite, char Freifläche, char Spieler);


/**
 * Kontrolle ob bewegung möglich -> ausserhalb der Map (return 0)
 * Kontrolle ob bewegung möglich -> Hindernis (return 1)
 * Kontrolle ob Spieler auf Schatz ist (return 2)
 * Map updaten - ohne Ausgabe (return 3)
 *
 * @param Höhe                  Grösse des Arrays - map[Höhe][Breite]
 * @param Breite                Anz. Zeichen im Array - map[Höhe][Breite]
 *
 * @param Spieler_alt_Höhe      Spielerposition vor der Bewegung
 * @param Spieler_alt_Breite    Spielerposition vor der Bewegung
 * @param Spieler_neu_Höhe      Spielerposition nach der Bewegung
 * @param Spieler_neu_Breite    Spielerposition nach der Bewegung
 *
 * @param Hindernis             Inhalt für die alte Position (Hindernis)
 * @param Freifläche            Inhalt für die alte Position (Freifläche)
 * @param Spieler               Inhalt für die neue Position (Spieler)
 * @param Schatz                Inhalt für die neue Position (Schatz)
 */
int checkMap(int Höhe, int Breite, int Spieler_alt_Höhe, int Spieler_alt_Breite,
int Spieler_neu_Höhe, int Spieler_neu_Breite, char Hindernis, char Freifläche, char
Spieler, char Schatz);

#endif // MAP_H ENDE
```

7.2.3 Map.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "Map.h"

// Globale Variable für die Map (Pointer auf Pointer)
char** map = NULL;

// Initialisiert das Map Modul (Zufallsgenerator)
void initMapModule() {
    srand(time(NULL));
}
```

```
// Speicher für die Map allozieren
void allocateMap(int Höhe, int Breite) {
    map = (char**)malloc(Höhe * sizeof(char*));          // Speicher für Zeilen
reservieren
    for (int i = 0; i < Höhe; i++) {
        map[i] = (char*)malloc(Breite * sizeof(char));   // Speicher für Spalten
reservieren
    }
}

// Speicher für die Map freigeben
void freeMap(int Höhe) {
    if (map != NULL) {
        for (int i = 0; i < Höhe; i++) {
            free(map[i]);                                // Speicher für jede Zeile
freigeben
        }
        free(map);                                       // Speicher für die
Zeilenzeiger freigeben
        map = NULL;                                      // Pointer zurücksetzen
    }
}

// Routinen zum erstellen der Map
void createMap(int Höhe, int Breite, char Hindernis, char Freiflaeche, int
Werscheinlichkeit) {

    for (int h = 0; h < Höhe; h++) {                     // Schleife für die
Höhe
        for (int b = 0; b < Breite; b++) {               // Schleife für die Breite
            if (rand() % 100 < Werscheinlichkeit) {      // Fülle die Map zu
Werscheinlichkeit % mit inhalt aus Hindernis, sonst aus Freiflaeche
                map[h][b] = Hindernis;
            } else {
                map[h][b] = Freiflaeche;
            }
        }
    }
}

// Routinen zum plazieren des Spielers - Rückgabe der Position des Spielers
Position placePlayer(int Höhe, int Breite, char Freiflaeche, char Spielers) {

    int plaziert = 0;                                    // Marker ob plaziert
    Position pos;                                        // Position des Spielers
initialisieren

    while (!plaziert) {                                  // Spieler
plaziert?
```

```
        int h = rand() % (Höhe);                        // Zufallszahl für Höhe
        int b = rand() % (Breite);                      // Zufallszahl für Breite
        if (map[h][b] == Freiflaeche) {                 // Nur belegen wenn inhalt
aus Position = char Freiflaeche
            map[h][b] = Spielers;                       // Spieler plazieren

            pos.x = h;                                  // Position speichern
            pos.y = b;

            plaziert = 1;                               // Schleife beenden

        }
    }
    return pos;
}

// Routinen zum plazieren des Schatzes - Rückgabe der Position des Schatzes
Position placeSchatz(int Höhe, int Breite, char Hindernis, char Schatz, char
Spielers) {

    int plaziert = 0;                                   // Marker ob plaziert
    Position pos;                                       // Position des Schatzes
initialisieren

    while (!plaziert) {                                 // Spieler
plaziert?
        int h = rand() % (Höhe);                        // Zufallszahl für Höhe
        int b = rand() % (Breite);                      // Zufallszahl für Breite
        if (map[h][b] != Hindernis && map[h][b] != Spielers) { // Nur belegen wenn
nicht Buchstabe i oder m
            map[h][b] = Schatz;                         // Schatz plazieren

            pos.x = h;                                  // Position speichern
            pos.y = b;

            plaziert = 1;                               // Schleife beenden
        }
    }
    return pos;
}

// Routinen zum ausgeben der Map
void printMap(int Höhe, int Breite) {
    printf("\n");                                       // Leere Zeile vor der
Map
    for (int i = 0; i < Höhe; i++) {                    // Zeilen Schleife
        for (int j = 0; j < Breite; j++) {              // Spalten Schleife
            printf("%c ", map[i][j]);                   // Ausgabe Zeichen
        }
```

```
    printf("\n");                                          // Neue Zeile beginnen
    }
    printf("\n");                                          // Leere Zeile nach der Map
}

// Routinen zum updaten der Map (Bewegung des Spielers)
void updateMap(int Spieler_alt_Höhe, int Spieler_alt_Breite, int Spieler_neu_Höhe,
int Spieler_neu_Breite, char Freifläche, char Spieler) {
    map[Spieler_alt_Höhe][Spieler_alt_Breite] = Freifläche;     // Alte Position
mit char Freifläche füllen
    map[Spieler_neu_Höhe][Spieler_neu_Breite] = Spieler;        // Neue Position
mit char Spieler füllen
}

// Kontrolliere/Mache die Bewegung des Spielers
int checkMap(int Höhe, int Breite, int Spieler_alt_Höhe, int Spieler_alt_Breite,
int Spieler_neu_Höhe, int Spieler_neu_Breite, char Hindernis, char Freifläche, char
Spieler, char Schatz) {
    if (Spieler_neu_Höhe < 0 || Spieler_neu_Höhe >= Höhe || Spieler_neu_Breite < 0
|| Spieler_neu_Breite >= Breite) {                  // Ausserhalb der Map?
        return 0;                                                       // 0 ==
Bewegung nicht möglich Ausserhalb
    }
    else if (map[Spieler_neu_Höhe][Spieler_neu_Breite] == Hindernis) {  // Auf
Hindernis?
        return 1;                                                       // 1 ==
Bewegung nicht möglich Hindernis
    }
    else if (map[Spieler_neu_Höhe][Spieler_neu_Breite] == Schatz) {     // Auf
Schatz? -> Map updaten
        updateMap( Spieler_alt_Höhe, Spieler_alt_Breite, Spieler_neu_Höhe,
Spieler_neu_Breite, Freifläche, Spieler);
        return 2;                                                       // 2 ==
Spiel gewonnen
    }
    else {                                                              // Kein
Problem? -> Map updaten
        updateMap( Spieler_alt_Höhe, Spieler_alt_Breite, Spieler_neu_Höhe,
Spieler_neu_Breite, Freifläche, Spieler);
        return 3;                                                       // 3 ==
Bewegung erfolgreich
    }
}
```

8