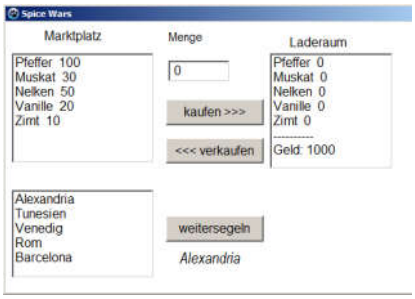
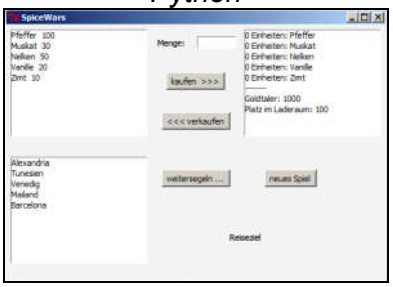
Zum Spiel

SpiceWars ist ein *Handelssimulationsspiel*. Der Spieler befindet sich im fiktiven 18. Jhdt und betreibt ein kleines Handelsunternehmen. Mit einer Kogge segelt er zwischen einigen Städten/Orten hin und her, kauft Gewürze an einem Ort und verkauft diese wieder woanders. Nach jeder Segeltour fallen und steigen die Preise. Spielgeschehen und Spielorte sind nicht historisch nachempfunden.

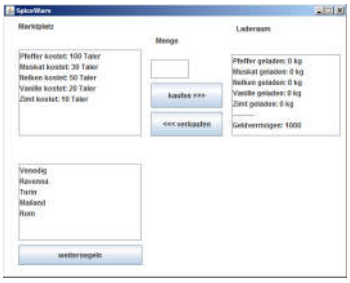
Lazarus



Python



Java



Die Grundversion des Spiels orientiert sich an Handelsspielen wie "Hanse" oder "Kaiser" und ordnet sich in die Liste der "rundenbasierten Spiele" ein. Während die Welle der Handelsspiele in den 90er Jahren mittlerweile abgeebbt ist, erleben Spiele dieser Art auf mobilen Plattformen wie Android ihre Renaissance.

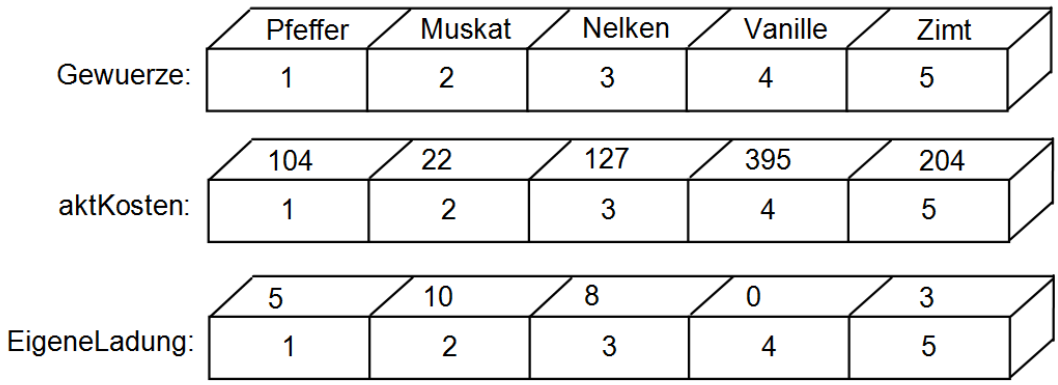
Oftmals handelt es sich allerdings um Spiele mit antisozialem Hintergrund, wie z.B. "DopeWars", das sich von der MS DOS Version bis hin auf die Android-Plattform über die Jahre gerettet hat. Als Drogendealer schlägt sich der Spieler dabei durch den Untergrund New Yorks. An dieser Stelle muss (auch im Unterricht) deutlich darauf hingewiesen werden, dass Spiele mit antigesellschaftlichem, drogen- oder gewaltverherrlichendem Hintergrund nicht tolerabel und abzulehnen sind.

Algorithmischer Kern

- Zugriff auf Daten in Arrays
- Arrays für verschiedene Datentypen (Integer, String)

Schritt 1: Die Daten und Anzeige der Daten

Datenhaltung im Array



Die Daten (Gewürze, Preise und Ladungsmenge) werden in Arrays abgespeichert. Die entsprechenden Indizes markieren ein Gewürz.
Damit ein Gewürz bequem mit der Maus auswählbar ist, und die maximal zu kaufende und verkaufende Menge automatisch berechnet und angezeigt wird, wird zur Darstellung der Arrayinhalte eine Listen anzeigende visuelle Komponente gewählt (Lazarus: ListBox / Java: JList / Python: Listbox), die den Index des gewählten Gewürzes auslesen lässt.

Die Größe des Laderaums (Mengenbegrenzung) und das zur Verfügung stehende Geld lässt sich über eine Integer-Variable nachhalten.

Das Anzeigen der Daten regelt eine Prozedur. Eine Anfangsbelegung der Daten erfolgt im Konstruktor.

Lazarus

```
var
  Geld: integer;
  Laderaum: integer;
  Gewuerze : Array [1..5] of String;
  aktKosten : Array [1..5] of Integer;
  EigeneLadung: Array [1..5] of Integer;



procedure TForm1.FormCreate(Sender: TObject);
begin
  Geld := 1000;
  Laderaum := 100;

  Gewuerze[1] := 'Pfeffer'; Gewuerze[2] := 'Muskat';
  Gewuerze[3] := 'Nelken'; Gewuerze [4]:= 'Vanille';
  Gewuerze[5] := 'Zimt';

  aktKosten[1] := 100; aktKosten[2] := 30;
  aktKosten[3] := 50; aktKosten[4] := 20;
  aktKosten[5] := 10;

  EigeneLadung[1] := 0;  EigeneLadung[2] := 0;  EigeneLadung[3] := 0;
  EigeneLadung[4] := 0;  EigeneLadung[5] := 0;

  // ListBox der Reiseziele
  Listbox3.Items.append('Alexandria');
  Listbox3.Items.append('Tunesien');
  Listbox3.Items.append('Venedig');
  Listbox3.Items.append('Rom');
  Listbox3.Items.append('Barcelona');
  Datenanzeigen;
end;



procedure DatenAnzeigen;
var i : integer;
begin
  Form1.ListBox1.clear;
  Form1.ListBox2.clear;
  for i := 1 to 5 do
      Form1.ListBox1.Items.Append(Gewuerze[i] + '  '
            + IntToStr(aktKosten[i]));
  for i := 1 to 5 do
         Form1.ListBox2.Items.Append(Gewuerze[i] + '  '
         + IntToStr(EigeneLadung[i]));
  Form1.Listbox2.Items.append('----------');
  Form1.ListBox2.Items.append('Geld: ' + IntToStr(Geld));
end;
```

Python

```
import tkinter, random, math

Geld = 1000
Laderaum = 100
Gewuerze = ["Pfeffer", "Muskat", "Nelken", "Vanille", "Zimt"]
aktKosten = [100, 30, 50 , 20 ,10]
EigeneLadung = [0,0,0,0,0]



def DisplayAktualisieren():
  global Gewuerze
  global aktKosten
  global EigeneLadung
  global Geld
  global Laderaum
  Liste.delete("0","end")
  Liste.insert("end", Gewuerze[0] + "  " + str(aktKosten[0]))
  Liste.insert("end", Gewuerze[1] + "  " + str(aktKosten[1]))
  Liste.insert("end", Gewuerze[2] + "  " + str(aktKosten[2]))
  Liste.insert("end", Gewuerze[3] + "  " + str(aktKosten[3]))
  Liste.insert("end", Gewuerze[4] + "  " + str(aktKosten[4]))

  ListeLaderaum.delete("0","end")
  ListeLaderaum.insert("end", str(EigeneLadung[0]) +
      ← " Einheiten: " + str(Gewuerze[0]))
  ListeLaderaum.insert("end", str(EigeneLadung[1]) +
      ←  " Einheiten: " + str(Gewuerze[1]))
  ListeLaderaum.insert("end", str(EigeneLadung[2]) +
      ←  " Einheiten: " + str(Gewuerze[2]))
  ListeLaderaum.insert("end", str(EigeneLadung[3]) +
      ←  " Einheiten: " + str(Gewuerze[3]))
  ListeLaderaum.insert("end", str(EigeneLadung[4]) +
      ←  " Einheiten: " + str(Gewuerze[4]))
  ListeLaderaum.insert("end", "-------")
  ListeLaderaum.insert("end", "Goldtaler: " + str(Geld))
  ListeLaderaum.insert("end", "Platz im Laderaum: " + str(Laderaum))

  ListeStaedte.delete("0","end")
  ListeStaedte.insert("end", "Alexandria")
  ListeStaedte.insert("end", "Tunesien")
  ListeStaedte.insert("end", "Venedig")
  ListeStaedte.insert("end", "Mailand")
  ListeStaedte.insert("end", "Barcelona")


def NeuesSpiel():
  global aktKosten
  global EigeneLadung
  global Geld
  global Laderaum
  aktKosten = [100, 30, 50 , 20 ,10]
  EigeneLadung = [0,0,0,0,0]
  Geld = 1000
  Laderaum = 100
  DisplayAktualisieren()

#GUI ---------------------------------
#Hauptfenster
Fenster = tkinter.Tk()
Fenster.title("SpiceWars")

#-------------------------
```

```
# Die kursiven, unterstrichenen Ereignisverknüpfungen müssen ohne
#Prozedur erstmal noch rausgelassen werden
#-------------------------
Liste = tkinter.Listbox (width=30, height = 10)
Liste.grid(padx = 5, pady = 5, row = 1, column = 1,
     ← columnspan = 1, rowspan=3)
Liste.bind('<<ListboxSelect>>', Liste1Click)

LabelMenge = tkinter.Label(Fenster, text = 'Menge: ')
LabelMenge.grid (padx = 5, pady=5, row = 1, column = 2)

EingabeMenge = tkinter.Entry(Fenster, width = 8)
EingabeMenge.grid(padx = 5, pady = 5, row = 1, column = 3)

ButtonKaufen = tkinter.Button
   ← (Fenster, text=' kaufen  >>> ', command = kaufen)

ButtonKaufen.grid(padx=5, pady = 5, row =2, column = 2, columnspan=2)
ButtonVerkaufen = tkinter.Button
   ← (Fenster, text=' <<< verkaufen ', command = verkaufen)

ButtonVerkaufen.grid(padx=5, pady = 5, row =3, column = 2, columnspan=2)

ListeLaderaum = tkinter.Listbox (width=30, height = 10)
ListeLaderaum.grid(padx = 5, pady = 5, row = 1, column = 5,
   ← columnspan = 2, rowspan=3)

ListeLaderaum.bind('<<ListboxSelect>>', Liste2Click)


#-----------------------
ListeStaedte = tkinter.Listbox (width=30, height = 10)
ListeStaedte.grid(padx = 5, pady = 20, row = 4, column = 1,
columnspan = 1, rowspan=3)

ButtonBewegen = tkinter.Button(Fenster, text = ' weitersegeln ... ',
   ←  command = Weitersegeln)

ButtonBewegen.grid(row=4, column=2, padx=5, pady=25, columnspan = 2)

LabelReiseziel = tkinter.Label(Fenster, text = 'Reiseziel ')
LabelReiseziel.grid (padx = 5, pady=5, row = 5, column = 2,
  ← columnspan = 4)

ButtonNeustart = tkinter.Button(Fenster, text = ' neues Spiel ',
   ← command = NeuesSpiel)

ButtonNeustart.grid(row=4, column=5, padx=5, pady=25)

NeuesSpiel()
# ---------------------
Fenster.mainloop()
```

Java

```
import java.math.*;
import java.util.Random;

public class SpiceWars extends JFrame {
  int Geld;
  int Laderaum;
  String [] Gewuerze = new String [5];
  int [] aktKosten = new int [5];
  int [] EigeneLadung = new int [5];

  Random rand = new Random();
```

```
public SpiceWars(String title) {
  super(title);

  //Variablen
  Geld = 1000;
  Laderaum = 100;
  Gewuerze[0] = "Pfeffer"; Gewuerze[1] = "Muskat";
  Gewuerze[2] = "Nelken"; Gewuerze [3] = "Vanille";
  Gewuerze[4] = "Zimt";

  aktKosten[0] = 100; aktKosten[1] = 30;
  aktKosten[2] = 50; aktKosten[3] = 20;
  aktKosten[4] = 10;

  EigeneLadung[0] = 0;  EigeneLadung[1] = 0;
  EigeneLadung[2] = 0;  EigeneLadung[3] = 0;
  EigeneLadung[4] = 0;
  [ . . .]
  }

public void DatenAnzeigen()  {
  jList1Model.removeAllElements();
  jList2Model.removeAllElements();
  jList3Model.removeAllElements();
  for (int i=0; i<5; i++ ) {
    jList1Model.addElement(Gewuerze[i] + " kostet: " +
         aktKosten[i] + " Taler");
  } // end of for

  for (int i=0; i<5 ;i++ ) {
    jList2Model.addElement(Gewuerze[i] + " geladen: " +
         EigeneLadung[i] + " kg");
  } // end of for

  jList2Model.addElement("---------");
  jList2Model.addElement("Geldvermögen: " + Geld);

  jList3Model.addElement("Venedig");
  jList3Model.addElement("Ravenna");
  jList3Model.addElement("Turin");
  jList3Model.addElement("Mailand");
  jList3Model.addElement("Rom");
}
```

Schritt 2: Kaufs- und Verkaufsmenge wählen

Die Menge der zu kaufenden oder zu verkaufenden Güter wird in einem "Edit-Feld" angezeigt. Um die Bedienung komfortabel zu gestalten, soll beim Klicken auf ein Gewürz gleich die maximal kaufbare bzw. verkaufbare Menge in dieses Feld eingetragen werden.

Lazarus

```
// ListBox - Gewürzanzeige
procedure TForm1.ListBox1Click(Sender: TObject);
var index : Integer;
begin
    index := ListBox1.ItemIndex+1;
    Edit1.Text:= IntToStr(floor(Geld/aktKosten[index]));
end;

// ListBox - Laderaumanzeige
procedure TForm1.ListBox2Click(Sender: TObject);
var index : Integer;
begin
```

	`index := ListBox2.ItemIndex+1;`
	`Edit1.Text:= IntToStr(EigeneLadung[index]);`
	`end;`

Python	`# Die Listenauswahl ist ein Tupel (für Mehrfachauswahl),`
	`# aus dem des erste Element (index) ausgelesen werden muss`
	`def Liste1Click(event):`
	`  Tupel = Liste.curselection()`
	`  index = int(Tupel[0])`
	`  EingabeMenge.delete(0, "end")`
	`  EingabeMenge.insert(0, math.floor(Geld/aktKosten[index]))`
	
	`def Liste2Click(event):`
	`  Tupel = ListeLaderaum.curselection()`
	`  index = int(Tupel[0])`
	`  if (index < 4):`
	`  EingabeMenge.delete(0, "end")`
	`  EingabeMenge.insert(0, EigeneLadung[index])`

Java	`// Liste Gewürzanzeige`
	`public void jList1_MouseClicked(MouseEvent evt) {`
	`  int index = jList1.getSelectedIndex();`
	`  jNumberField1.setText(Integer.toString((Geld / aktKosten[index])));`
	`  }`
	
	`// Liste Laderaum`
	`  public void jList2_MouseClicked(MouseEvent evt) {`
	`  int index = jList2.getSelectedIndex();`
	`  jNumberField1.setText(Integer.toString (EigeneLadung[index]));`
	`  }`

Schritt 3 – Kaufen und Verkaufen

Beim Kauf wird dabei die Anzahl der Gewürzeinheiten mit den Kosten multipliziert und geprüft, ob das Geld ausreicht und ob genug Laderaum zur Verfügung steht. Beim Verkauf wird geprüft, ob die zu verkaufende Menge im Laderaum überhaupt vorhanden ist.

Lazarus	`// KaufButton`
	`procedure TForm1.Button1Click(Sender: TObject);`
	`var Anzahl : integer;`
	`  Nummer : integer;`
	`begin`
	`  Anzahl := StrToInt(Edit1.text);`
	`  Nummer := ListBox1.ItemIndex+1;`
	`  if Anzahl*aktKosten[Nummer] <= Geld then`
	`  begin`
	`  Geld := Geld - Anzahl * aktKosten[Nummer];`
	`  EigeneLadung[Nummer] := EigeneLadung[Nummer]+Anzahl;`
	`  end;`
	`  DatenAnzeigen;`
	`end;`
	
	`// VerkaufsButton`
	`procedure TForm1.Button2Click(Sender: TObject);`
	`var Anzahl : integer;`
	`  Nummer : integer;`
	`begin`
	`  Anzahl := StrToInt(Edit1.text);`
	`  Nummer := ListBox2.ItemIndex+1;`
	`  if Anzahl <= EigeneLadung[Nummer] then`
	`  begin`

```
        Geld := Geld + Anzahl * aktKosten[Nummer];
        EigeneLadung[Nummer] := EigeneLadung[Nummer]-Anzahl;
        end;
    DatenAnzeigen;
end;
```

Python

```
def kaufen():
  global Geld
  global aktKosten
  global EigeneLadung
  global Laderaum
  Anzahl = int(EingabeMenge.get())
  Nummer = int(Liste.curselection()[0])
  if (Laderaum >= Anzahl) and (Geld >= aktKosten[Nummer]*Anzahl):
      Laderaum = Laderaum - Anzahl
      Geld = Geld - int(aktKosten[Nummer])*Anzahl
      EigeneLadung[Nummer] = EigeneLadung[Nummer]+Anzahl
  DisplayAktualisieren()


def verkaufen():
  global Geld
  global aktKosten
  global EigeneLadung
  global ListeLaderaum
  global Laderaum
  Anzahl = int(EingabeMenge.get())
  Nummer = int(ListeLaderaum.curselection()[0])
  if (Anzahl <= EigeneLadung[Nummer]):
      Laderaum = Laderaum + Anzahl
      Geld = Geld + int(aktKosten[Nummer])*Anzahl
      EigeneLadung[Nummer] = EigeneLadung[Nummer] - Anzahl
  DisplayAktualisieren()
```

Java

```
  // KaufButton
  public void jButton1_ActionPerformed(ActionEvent evt) {
    int Anzahl = Integer.parseInt(jNumberField1.getText());
    int Nummer = jList1.getSelectedIndex();
    if (Anzahl*aktKosten[Nummer] <= Geld) {

        Geld = Geld - Anzahl * aktKosten[Nummer];
        EigeneLadung[Nummer] = EigeneLadung[Nummer]+Anzahl;
    }
    DatenAnzeigen();
  }

  // VerkaufsButton
  public void jButton2_ActionPerformed(ActionEvent evt) {
    int Anzahl = Integer.parseInt(jNumberField1.getText());
    int Nummer = jList2.getSelectedIndex();
    if (Anzahl <= EigeneLadung[Nummer]) {
        Geld = Geld + Anzahl * aktKosten[Nummer];
        EigeneLadung[Nummer] = EigeneLadung[Nummer]-Anzahl;
    }
    DatenAnzeigen();
  }
```

4. Schritt - Weitersegeln

Beim "Weitersegeln" zu einer neuen Stadt ändern sich die Preise. Tatsächlich ist es jedoch egal, wo der Spieler hin segelt, da die Preise unabhängig von einem Ort zufällig ermittelt werden. (Wäre das nicht so, würde ein Spieler das Gewürzgefälle eines Gewürzes zwischen zwei Orten ausnutzten und nur mit einem Gewürz in zwei verschiedenen Orten handeln.)

Die neuen Gewürzkosten richten sich nach einem vorher festgelegten Maximal bzw. Minimalwert.

Lazarus

```
var
  [. . .]
  KostenMin : Array [1..5] of Integer;
  KostenMax : Array [1..5] of Integer;

procedure TForm1.FormCreate(Sender: TObject);
begin
[. . . ]
  KostenMin[1]:=50; KostenMin[2]:=5; KostenMin[3]:=20;
  KostenMin[4]:= 100; KostenMin[5]:= 60;

  KostenMax[1]:=150; KostenMax[2]:=35; KostenMax[3]:=170;
  KostenMax[4]:= 450; KostenMax[5]:= 300;
end;

// WeitersegelButton
procedure TForm1.Button3Click(Sender: TObject);
var i : integer;
begin
  Label2.caption := Listbox3.GetSelectedText;
  for i := 1 to 5 do
     begin
       aktKosten[i] := KostenMin[i]+random(KostenMax[i]-KostenMin[i]);
     end;
  DatenAnzeigen;
end;
```

Python

```
KostenMin = [50, 5 , 20 , 100, 60]
KostenMax = [150, 35 , 170 , 450, 300]


def Weitersegeln():
  global ListeStaedte
  global aktKosten
  global KostenMin
  global KostenMax
  Tupel = ListeStaedte.curselection()
  index = int(Tupel[0])
  if (index != 0):
    Stadt = ListeStaedte.get(index)
    LabelReiseziel.config(text="Ihre Reise geht nach " + Stadt
        ←   + ".\n Der Wind steht gut.\n Sie brauchen 2 Wochen")
    for i in range (5):
       aktKosten[i] =  random.randint(KostenMin[i],KostenMax[i])
    DisplayAktualisieren()
```

Java

```
public class SpiceWars extends JFrame {
  [. . .]
  int [] KostenMin = new int [5];
  int [] KostenMax = new int [5];

  public SpiceWars(String title) {
```

```
   [ . . .]
   KostenMin[0] = 50; KostenMin[1]= 5;
   KostenMin[2] = 20;  KostenMin[3] = 100;
   KostenMin[4] = 60;

   KostenMax[0] = 150; KostenMax[1] = 35;
   KostenMax[2] = 170;  KostenMax[3] = 450;
   KostenMax[4] = 300;


// WeitersegelButton
public void jButton3_ActionPerformed(ActionEvent evt) {
    for (int i = 0; i<5 ;i++ ) {
    aktKosten[i] = KostenMin[i]+
          rand.nextInt(KostenMax[i]-KostenMin[i]);
    }
    DatenAnzeigen();
}
```

Erweiterungen

Das Spiel als solches ist spielbar, es fehlen aber noch einige grundlegende Spielelemente, um das Spiel tatsächlich "dauerhaft spannend" zu halten.

Mögliche Erweiterungen:

1) Spielrunden! Das Spiel sollte nicht ewig dauern, sondern nach einer gewissen Rundenzahl (Zeit) beendet sein. Ein guter Zeitraum wäre ein Jahr. Wenn ein Ortswechsel 2 Wochen dauert, würde das Spiel nach 26 Spielrunden enden.

2) Schulden! Der Spieler startet neben einem Startkapital auch mit einigen Schulden, die pro Spielzug um ca. 10% erhöht werden und während des Spiels zurückgezahlt werden könnten. Sind die Schulden nach Beenden des Spiels nicht getilgt, verliert der Spieler.

3) Ereignisse! Bei einem Spielzug könnte zufällig ein Ereignis passieren, das die Preise für ein Gewürz beim nächsten Zug extrem steigen oder fallen lässt. *("Ein Sturm hat die Handelsflotte der Konkurrenz für 2 Wochen im Hafen gehalten. Die Preise für Pfeffer und Vanille verdreifachen sich")* oder *("Eine neue Handelsroute wurde erschlossen. Die Preise für Muskat fallen um 70%.")*

4) Platzprobleme! Nach einigen Runden wird der Laderaum mit 100 Einheiten zu klein. Dem Spieler könnte in regelmäßigen Abständen ein weiteres Schiff mit 100 Einheiten zu einem gewissen Preis angeboten werden. Der Laderaum würde sich damit um z.B. 100 Einheiten vergrößern.

5) Piraten! Auch Piraten könnten das Schiff überfallen und die gesamte Ladung rauben. Dem Spieler könnten Soldaten als Begleitung für die Handelsflotte angeboten werden. Pro Soldat könnte sich die Chance, die Piraten zu besiegen um 25% erhöhen. 4 Soldaten würden demnach 100% Sicherheit vor Piraten bieten.

Vorlage für eine Schülerdokumentation

Erweitere das Spiel um einige Features und **erstelle eine Dokumentation** über die Erweiterungen. Die Dokumentation wird als Text getippt, abgegeben und mit dem Spiel benotet.

Das *Deckblatt der Dokumentation* sieht wie folgt aus (Formatierung hier ist jedem selbst überlassen):

Name der Schule
Bezeichnung des Kurses

Dokumentation der Erweiterungen zur
Handelssimulation "SpiceWars" (oder ähnlicher Titel)

Angefertigt von: *Name*
Datum: *Datum*
Kurs: *Kurs*

Inhalt

Die Dokumentation muss in folgende Punkte gegliedert sein:

1.) Liste mit allen Erweiterungen in verbaler umfassender Kurzbeschreibung ohne technische Umsetzung
2.) Darstellung der technischen Umsetzung einer Erweiterung
a) Beschreibung der technischen Umsetzung
b) alle Zeilen im groben Zusammenhang, die zur Erweiterung gehören
c) alle wesentlichen Sreenshots des laufenden Spiels, bei denen die Funktionalität der Erweiterung zu sehen ist

Die Dokumentation beginnt mit der 2. Seite, die Seiten sind nummeriert.

- Schriftgröße: 11 oder 12, Überschriften selbstständig fett und/oder evtl. größer
- Schriftart: Arial oder Times – Quelltexte: Courier New und 2 pt kleiner – Zitate: kleiner und kursiv und Blocksatz. Auslassungen mit [...] kennzeichnen – Bildbeschreibung: kursiv und kleiner unter das Bild
- Umfang: 3 - 5 Seiten, **ausgedruckt** und getackert, nicht geheftet. Keine E-Mail!
- Auf Formatierung des Textes, der Textblöcke und Bilder achten (Tabulator, Tabellen, etc.)!

Benotungsgrundlage (jeweils eine Note für Spiel und Dokumentation)

- Erweiterungen und Umsetzung des Spiels (technische und praktische Umsetzung)
- fachliche Erstellung des Programmcodes (Einrückung, gängige Programmierregeln, Lesbarkeit, etc.)
- Spielfreude, Spielbarkeit

- Klarheit, Sprache und Lesbarkeit der Dokumentation,
- Layout und Formatierung
- Informationsgehalt der Dokumentation