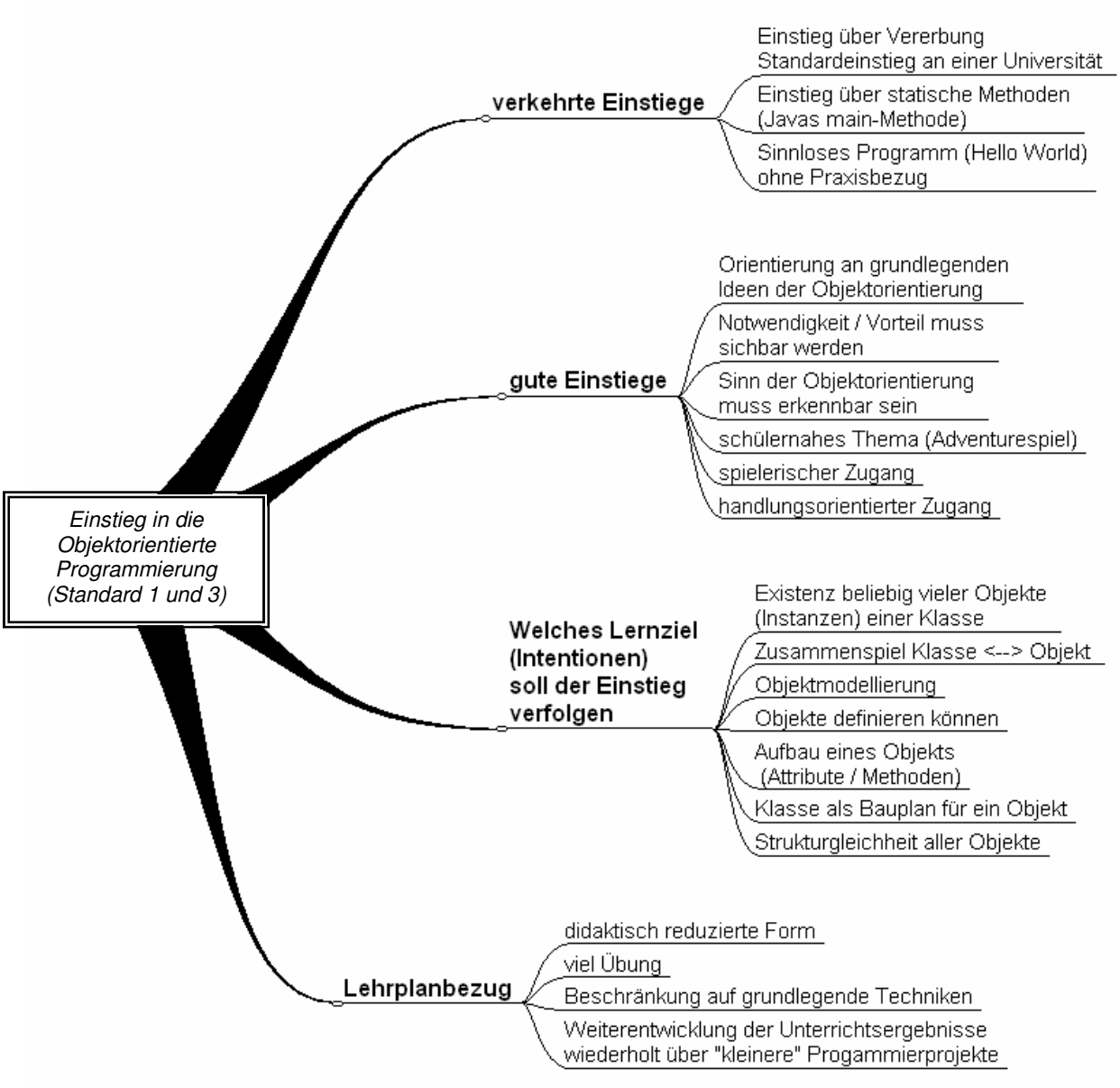
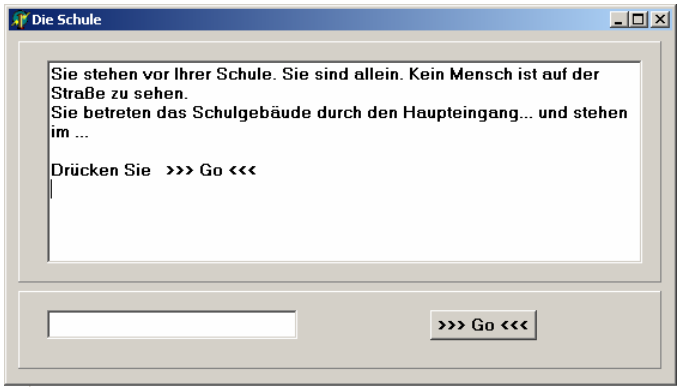
Ergebnisse der Seminarsitzung vom 19. November 2007



„Die Schule“ - ein Textadventure

a) Didaktische Überlegungen

Textadventures sind Computerspiele der „ersten Generation“, also Spiele, bei denen der Spieler nur über Texteingaben mit dem Computer kommunizieren kann. Seine Figur steuert er über Befehle wie „go“, „east“, „west“, „pick up“, „turn“, etc. Die ersten Textadventures waren Don Woods „Collossal Cave Adventure“ (1976) und das von MIT-Studenten programmierte “Zork” (1977). Der Einstieg in die objektorientierte Programmierung erfolgt über die Entwicklung eines solchen Textadventures.



Prinzipiell kann solch ein Textadventure auch anders modelliert werden, über das Modell des Zustandsdiagramms etwa und wäre damit auch als Einführungsbeispiel für einen allgemeinen Automaten denkbar. Die Implementierung erweist sich jedoch aufgrund der geringen Erweiterbarkeit gegenüber dem objektorientierten Ansatz als deutlich schwieriger und wenig motivierend.

Das Einstiegsbeispiel ist bewusst einfach gehalten, auf das Anlegen neuer Units für eine Klasse oder Datenkapselung (private/ public) wird bewusst verzichtet, um die zentralen Begriffe der Objektorientierung herauszuarbeiten. Auch ist hier von der Verwendung der set-/ get-Methoden für den Zugriff auf Attribute abgesehen. Auf eine Darstellung der Klassen im Klassendiagramm in UML-Notierung soll aufgrund des erhöhten Abstraktionsniveaus ebenfalls abgesehen werden.

Im Zentrum der Programmentwicklung stehen „lediglich“ die fundamentalen Begriffe:

- **Objekt / Klasse,**
- **Referenz,**
- **Klassendiagramm,**
- **Objektdiagramm.**

Ein besonderer Vorteil des Textadventures ist die mannigfache Möglichkeit der Erweiterbarkeit durch die Schüler[1]. Im Ergänzen von Räumen und Erweitern der Klassen durch Attribute üben sie den Umgang mit Objekten. Über die Ausgabe der Raumbeschreibung und referenzierten Raumobjekte fassen die Schüler den Zustand der Objekte.

Weitere Begriffe können in der Erweiterung des Spiels thematisiert werden. Denkbar ist z.B. das Thematisieren von ***Klassenattributen***, die einen Zeitfaktor darstellen können (nach 10 Raumwechseln werden Geräusche hörbar oder wird das Gebäude von der Polizei gestürmt o.ä.). Auch kann der ***Konstruktor*** der Klasse TRaum erweitert und thematisiert werden, so dass die Beschreibung eines Raumes gleich bei der Instantiierung des Raumobjekts übergeben wird.

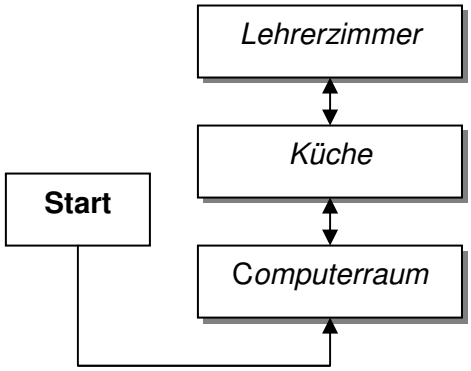
*Hinweis: **Klassenattribute** gibt es nur in Java oder ähnlichen Sprachen. In der benutzten Programmiersprache Delphi müssen sie durch globale Variablen oder ‚properties' nachgebildet werden.*

Nun können Sie Ihre Schüler erstmal spielen lassen!

[1] Schüler wird hier stets in seiner generischen Bedeutung verwendet.

b) Strukturanalyse

Zur Strukturbeschreibung soll ein Adventure mit drei Räumen dienen, die lediglich über die Zugänge *Norden* und *Sueden* begangen werden können.

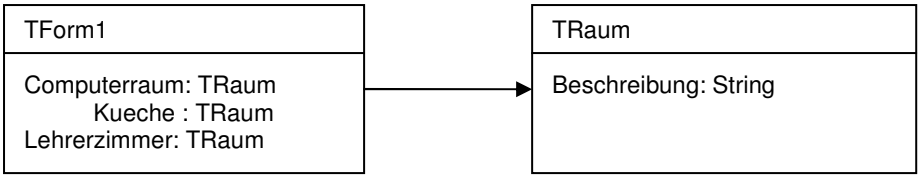


Zum Verständnis der Arbeitsweise wird das Programm in die Entwicklungsversionen 0.1, 0.2 und 1.0 unterteilt, die nacheinander beschrieben werden. Version 1.0 ist spielbar. Ab dieser Version sind die Schüler in der Lage, selbst ihre Programme zu entwickeln. Im Unterricht kann bis zur Version 1.0 „frontal" vorgegangen werden, da den Schülern im Anschluss an diese Frontalphase eine längere Kreativphase zur Erstellung der eigenen Programme eingeräumt wird.

Version 0.1

- Erstmal geht es um **Klassen** und **Attribute**. Es wird eine Klasse TRaum erstellt, die ein Attribut (Eigenschaft) *Beschreibung* vom Typ String hat.
- Beim Start des Programms werden drei Objekte vom Typ TRaum erzeugt.
- Über die ButtonClick-Methode werden die Beschreibungstexte der „Räume" je nach Eingabe des Raumes über ein Edit-Feld ausgegeben (Die Räume können in dieser Version noch nicht über die Himmelsrichtungen gewechselt werden).

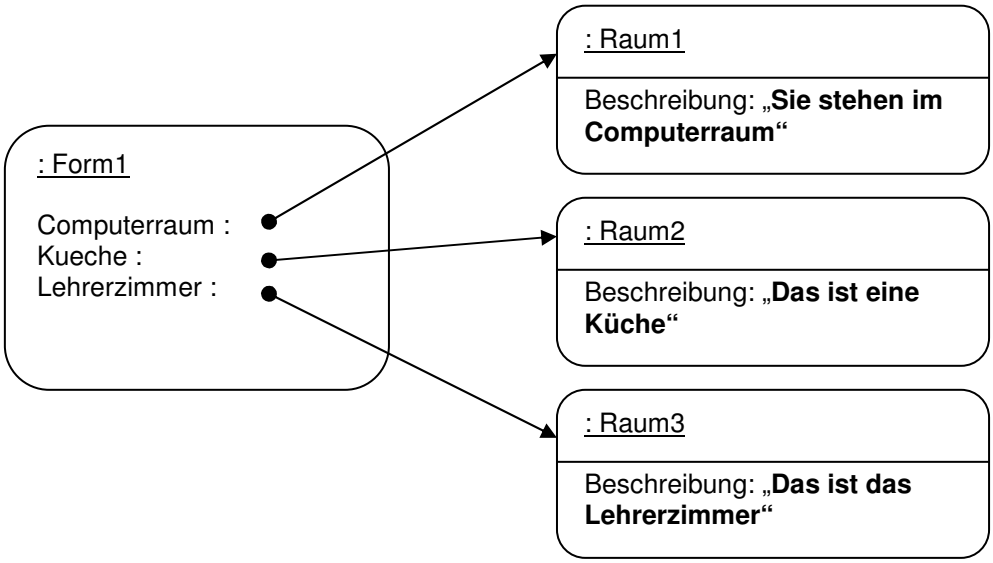
Klassendiagramm




```
type
  TRaum = class
     Beschreibung : String;
  end;

var
  Form1: TForm1;
  Computerraum, Kueche, Lehrerzimmer : TRaum;

implementation
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
Memo1.Lines.Add ('Hallo, willkommen im Textadventure');
Computerraum := TRaum.Create;
Kueche := TRaum.Create;
Lehrerzimmer := TRaum.Create;
```

```
Computerraum.Beschreibung := ('Sie stehen im Computerraum.');
Kueche.Beschreibung := ('Das ist eine Küche');
Lehrerzimmer.Beschreibung := ('Das ist das Lehrerzimmer');
end;

procedure TForm1.Button1Click(Sender: TObject);
  var Eingabe : String;
  begin
  Eingabe := Edit1.text;
  if Eingabe = 'Kueche' then  Memo1.Lines.add (Kueche.Beschreibung);
  if Eingabe = 'Computerraum' then  Memo1.Lines.add (Computerraum.Beschreibung);
  if Eingabe = 'Lehrerzimmer' then  Memo1.Lines.add (Lehrerzimmer.Beschreibung);
end;
```

Objektdiagramm



An dieser Stelle sollten noch einmal die **Grundbegriffe der objektorientierten Programmierung** mit dem dazugehörigen Delphicode thematisiert und wiederholt werden. Das ist deshalb so wichtig, weil die Schüler im übernächsten Schritt ihre Programme selbst erstellen und sich weniger mit Nachschlagen des entsprechenden Codes als mit kreativer Entwicklung ihrer Programme beschäftigen sollen.

Deklaration einer Klasse	*type* *TRaum = class* *Beschreibung : String;* *end;*
Deklaration von Attributen (Eigenschaften)	*Beschreibung : String;* *Computerraum : TRaum;*
Erzeugen von Objekten (Instanzen)	*Computerraum:= TRaum.Create;*
Zugriff auf die Attribute eines Objekts über eine Referenz (Zeiger auf ein Objekt)	*Computerraum.Beschreibung := ('Sie stehen im Computerraum.');* *Edit1.text := Computerraum.Beschreibung;*

Version 0.2

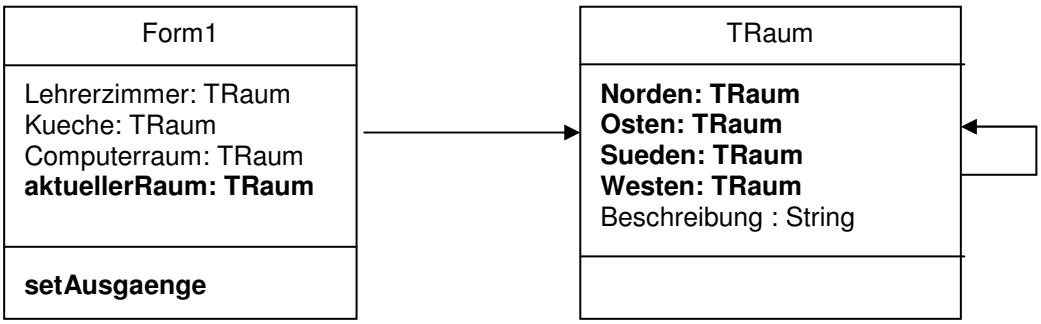
- Die Klasse TRaum wird um die **Attribute**, die mögliche **Referenzen auf benachbarte Räume** darstellen (Norden, Sueden, Westen, Osten) erweitert.

- Die Klasse TRaum wird um die Objektmethode **setAusgaenge** erweitert, in der die Referenzen übergeben werden.
- Mithilfe des Attributs *aktuellerRaum* als Referenz auf jeweils einen Raum können die Räume nun über die Himmelsrichtungen gewechselt werden.

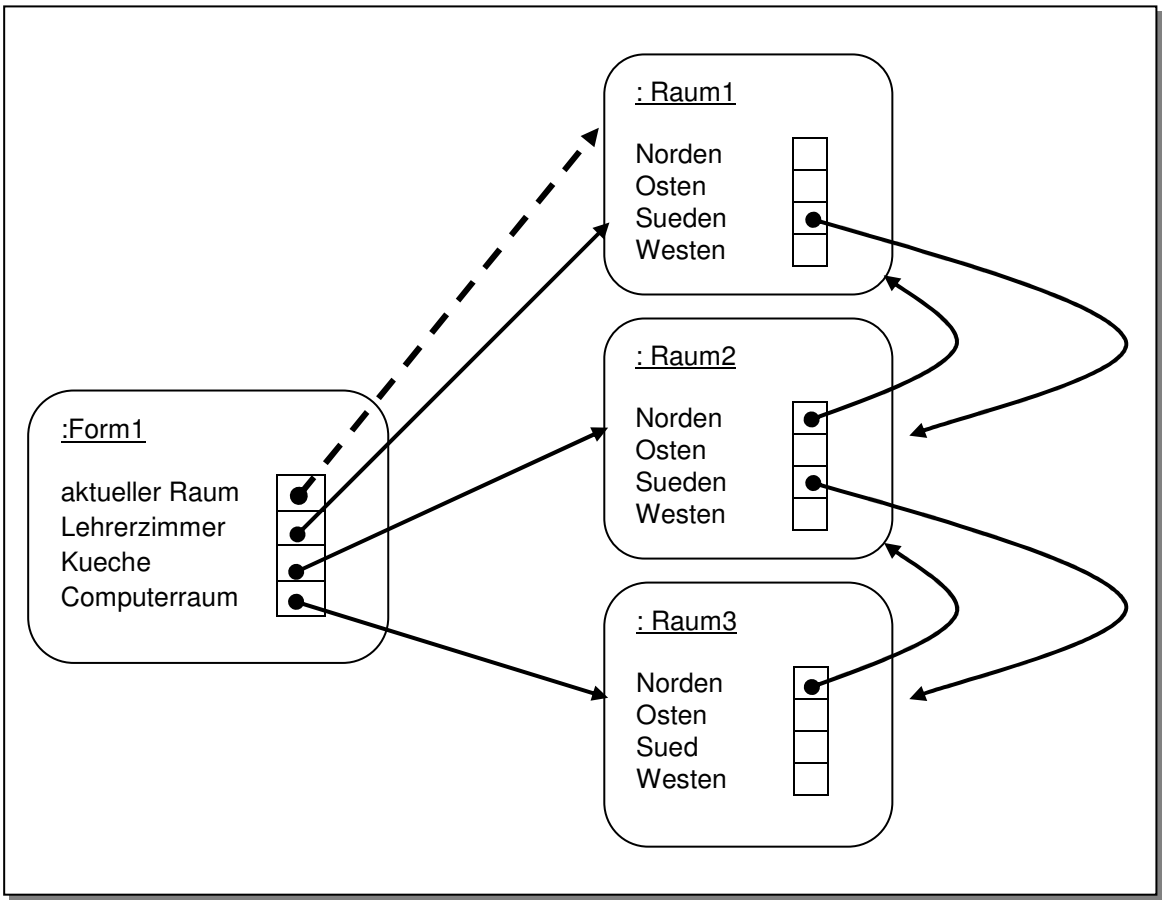
```
type
  TRaum = class
     Beschreibung : String;
     Norden, Sueden, Westen, Osten : TRaum;
     procedure setAusgaenge (norden, osten, sueden, westen : TRaum);
  end;

var
  Form1: TForm1;
  Computerraum, Lehrerzimmer, Kueche : TRaum;
  aktuellerRaum : TRaum;
implementation
{$R *.dfm}

procedure TRaum.setAusgaenge (norden, osten, sueden, westen :TRaum);
begin
  Norden := norden;
  Osten := osten;
  Sueden := sueden;
  Westen := westen;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
[…]
Computerraum.setAusgaenge(Kueche, nil, nil, nil);
Kueche.setAusgaenge(Lehrerzimmer, nil, Computerraum, nil);
Lehrerzimmer.setAusgaenge(nil, nil, Kueche, nil);
aktuellerRaum := Lehrerzimmer;
end;

procedure TForm1.Button1Click(Sender: TObject);
  var Eingabe : String;
  begin
  Eingabe := Edit1.text;
  // --- Raum wechseln ---
  if Eingabe = 'Norden' then aktuellerRaum := aktuellerRaum.Norden;
  if Eingabe = 'Osten' then aktuellerRaum := aktuellerRaum.Osten;
  if Eingabe = 'Sueden' then aktuellerRaum := aktuellerRaum.Sueden;
  if Eingabe = 'Westen' then aktuellerRaum := aktuellerRaum.Westen;
  // --- Beschreibung des aktuellen Raums ausgeben
  Memo1.Lines.add (aktuellerRaum.Beschreibung);
  end;
```

Das Klassendiagramm und das nach dem Programmstart entstandene Objektdiagramm können jetzt erweitert werden.

Klassendiagramm



Objektdiagramm

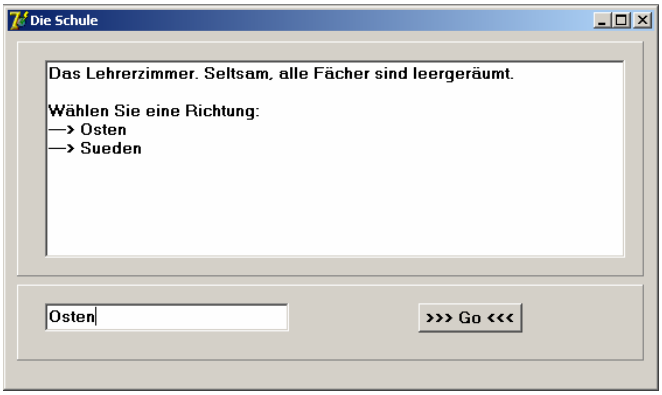


Version 1.0

- Mit einer Ausgabe beim Betreten eines Raumes, in welche Himmelsrichtungen der Raum wieder verlassen werden kann, wird das Programm spielbar.
- Der Test vor einem Raumwechsel, ob über die entsprechende Referenz ein Raumobjekt tatsächlich referenziert wird, schützt vor Abstürzen und Fehlermeldungen.

```
procedure TForm1.Button1Click(Sender: TObject);
  var Eingabe : String;
  begin
  Eingabe := Edit1.text;
  // --- Raum wechseln ---
  if (Eingabe = 'Norden') and (aktuellerRaum.Norden <> nil) then
       aktuellerRaum := aktuellerRaum.Norden;
  if (Eingabe = 'Osten') and (aktuellerRaum.Osten <> nil) then
       aktuellerRaum := aktuellerRaum.Osten;
  if (Eingabe = 'Sueden') and (aktuellerRaum.Sueden <> nil) then
       aktuellerRaum := aktuellerRaum.Sueden;
  if (Eingabe = 'Westen') and (aktuellerRaum.Norden <> nil) then
       aktuellerRaum := aktuellerRaum.Westen;
  // --- Beschreibung des aktuellen Raums ausgeben
  Memo1.Lines.add (aktuellerRaum.Beschreibung);
  // --- mögliche Ausgänge der Räume  angeben ---
  if aktuellerRaum.Norden <> nil then Memo1.Lines.add (' ---> Norden');
  if aktuellerRaum.Osten <> nil then Memo1.Lines.add (' ---> Osten');
  if aktuellerRaum.Sueden <> nil then Memo1.Lines.add (' ---> Sueden');
  if aktuellerRaum.Westen <> nil then Memo1.Lines.add (' ---> Westen');
  end;
```

Screenshot der Version 1.0



c) Schülerarbeiten und Erweiterungen

Mit der Version 1.0 können die Schüler nun weiterarbeiten, ein eigenes Spiel erstellen und ggfs. anschließend dokumentieren. Dabei sollen die Schüler zunächst möglichst frei von Programmcode ein eigenes Abenteuerspiel entwerfen, eine Vorgeschichte und Spielsituation erfinden, einen Schauplatz wählen, Räume zeichnen und sich eine Spielerfigur ausdenken. Je kreativer und unbefangener die Schüler hier ans Werk gehen, desto besser fallen erfahrungsgemäß die späteren Arbeitsergebnisse aus. Die Praxis mit Delphi hat hier gezeigt, dass letztlich fast alles, was Schüler umsetzen wollen, von Delphi unterstützt wird und schnell und leicht implementiert werden kann. Selbstredend sollte der Lehrer hier auch entsprechende Hilfestellungen geben.

Beliebte und mögliche Erweiterungen des Spiels sind etwa:

1. Bilder der Räume einblenden

Alle Bilder werden in einem Ordner ***Bilder*** im Projektordner abgelegt, z.B. das Bild ***Klassenraum.jpg,***
und der Ort des Bildes als *String-Variable* im Objekt der Klasse :

```
type TRaum = class
   Beschreibung : String;
   FotoReferenz : String;
   Norden, Westen, Sueden, Osten : TRaum;
   procedure setAusgaenge (N , O , S , W : Traum);
end;
```

abgelegt: ***Klassenraum.Fotoreferenz := 'Bilder/Klassenraum.jpg'.***
Das Foto wird von der Komponente ***Image*** im Attribut ***Picture*** beim Raumwechsel angezeigt.
Foto in des Objekt laden: ***Image1.Picture.LoadFromFile(aktuellerRaum.Fotoreferenz),***
Foto entfernen: ***Image1.Picture.CleanupInstance .***

2. Einen anderen Ausgabetext in einem schon besuchten Raum einblenden

TRaum erhält ein Attribut ***besucht*** (vor den Methodendeklarationen), das mit false initialisiert wird.
Gemäß ***besucht*** wird entweder die ***Beschreibung1*** oder die ***Beschreibung2*** ausgegeben:

```
type TRaum = class
   Beschreibung1 : String;
   Beschreibung2 : String;
   FotoReferenz : String;
   besucht : Boolean;
```

```
   Norden, Westen, Sueden, Osten : TRaum;
   procedure setAusgaenge (N , O , S , W : Traum);
end;
```

3. Weitere Ausgänge hinzufügen (Treppe rauf, Treppe runter)

Als weitere Ausgänge sind Treppen nach oben oder unten denkbar. Sie werden dann als weitere Ausgänge definiert, evtl. mit ***nil*** initiiert oder entsprechend verlinkt.

type TRaum = class
  Beschreibung : String;
  Norden, Westen, Sueden, Osten : TRaum;
  procedure setAusgaenge (N , O , S , W, **hoch, runter** : Traum);
end;

procedure TRaum.setAusgaenge(N, O , S, W, **hoch, runter** : TRaum);
begin
[…]
self.hoch := hoch;
self.runter := runter;
end;

In der Regel erweitern Schüler ihrer Programme noch um einen Rucksack (boolesche Variablen) um Items sammeln zu können und binden Sounds, Musik und Geräusche über den Media-Player ein.

Zwei Screenshots von fertigen und spielbaren Schülerarbeiten sollen hier nur gezeigt werden, um den Ideenreichtum von Schülern zu dokumentieren: