Technische Universität Darmstadt
Seminar Informatik SS 06
Prof. Dr. Fürnkranz

Seminar
„Knowledge Engineering und
Lernen in Spielen“

Pattern Movement

Ante Zubac

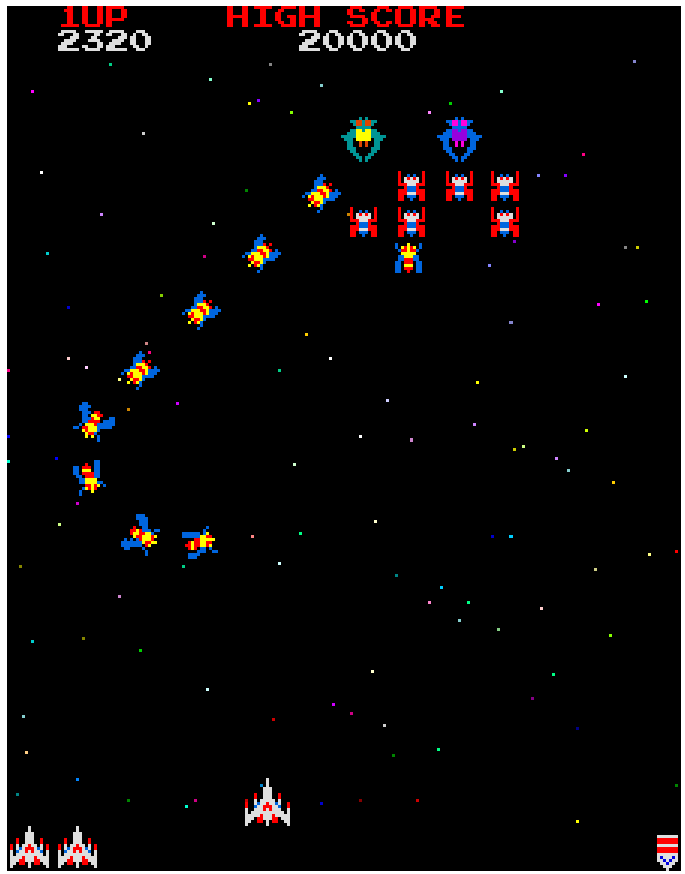
02.05.2006

Einleitung

- Pattern Movement
  - (Vordefinierte) Bewegungsmuster
    - Kreise, Quadrate, Schleifen, etc.
  - Vortäuschung intelligenten Verhaltens in Spielen für computer-gesteuerte Figuren
- Implementierung abhängig von Umgebung des Spiels
  - gerastert
  - kontinuierlich (z.B. First-Person Shooter)

Beispiel: Galaga

- Gegnerische Schiffe folgen Bewegungsmustern
- Abhängig von
  - Typ des Schiffes
  - Level
- Einfaches Beispiel
  - Bewegungen unabhängig von Aktionen des Spielers



Gliederung

- **Einleitung**
- **Spiele-Umgebungen**
- **Pattern Movements**
  - in gerasterten Umgebungen
  - in physikalisch simulierten Umgebungen
- **Vergleich**

Spiele-Umgebungen

Spiele-Umgebung gerastert

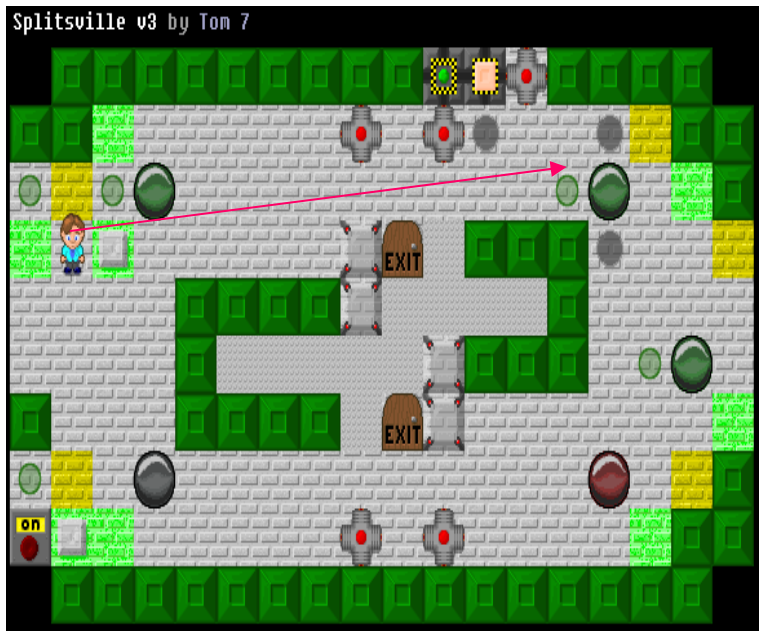
Spiel mit 18 x 10 Kacheln

Figur kann in 8 Richtungen bewegt werden

- Abbiegen 45° und 90°
- benachbarte Kacheln

Problem für computer-gesteuerte Figuren:
direkter Weg von A nach B?

Lösung:
Bresenham´s Algorithmus



„Splitsville“

Spiele-Umgebung physikalisch simuliert

- Kontinuierlicher Raum
- Physics Engine
  - führt Bewegungen aus unter Berücksichtigung aller Faktoren
- Höhere Intelligenzebene

„Age of Empires III“

Pattern Movement

Implementierung

- Standard-Vorgehen
  - **Control data** besteht aus Bewegungsbefehlen
    - z.B. *moveForward, moveBack, turnRight, turnLeft*
    - können als Gleitkommazahlen definiert werden
      - Bedeutung bei gerasterter / kontinuierlicher Umgebung?
    - andere Befehle möglich
      - *fireWeapon*, *dropBomb*
  - **Pattern Arrays**
    - gewünschtes Muster wird in Arrays gespeichert
    - jedes Feld enthält die Control Data Structure mit gewünschten Anweisungen

Beispiel (vereinfacht)

```
Pattern[0].moveForward = 5;
Pattern[1].turnRight = 1;
Pattern[2].moveForward = 5;
```

Game Loop

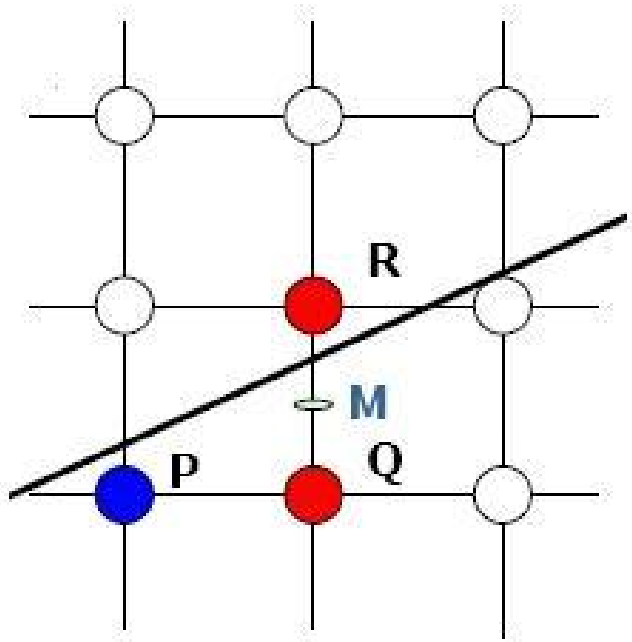
- Methode
- Taktgeber für die vordefinierten Pattern-Arrays
  - Instruktion, die dem aktuellen Zählerstand entspricht
- Inkrementiert den Zähler nach jeder erfolgten Instruktion
- Bewegungsfolgen werden sequentiell abgearbeitet

Implementierung in gerasterter Umgebung

- Bresenham´s line algorithm
  - Berechnet einen Weg von einem Startpunkt zu einem Zielpunkt
- Wird jetzt angewandt, um Bewegungsmuster zu erstellen
  - Durch Verbindung mehrerer line segments
  - Endpunkt A = Startpunkt B
  - Beispiel: Figur bewacht ein Tor und bewegt sich ständig hin und her
- Z.B. Rechteck-Muster
  - 4 line segments
  - Jede line wird durch Bresenham-Algorithmus berechnet

Bresenham´s Algorithmus
- Vorgehensweise

- Finde Pixel, der näher zum Mittelpunkt M liegt
  → Linie über oder unter M?
- Setze R falls Linie über M, ansonsten setze Q
- Iterativ weitermachen bis Endpunkt erreicht
- Unterschied jetzt
  - Mehrmaliges Anwenden
  - Line segments verbunden
  - Endpunkt A = Startpunkt B
  - Kontinuierliche Bewegung



Erstellung von Bewegungsmustern

- Punkte im Koordinatensystem
- Oben links: (0,0)
- BildeWegSegment (x_1, y_1, x_2, y_2);
  - Anwendung des Bresenham Algorithmus

- Beispiel Rechteckmuster
  - BildeWegSegment (10, 3, 18, 3);
  - BildeWegSegment (18, 3, 18, 12);
  - BildeWegSegment (18, 12, 10, 12);
  - BildeWegSegment (10, 12, 10, 3);

  - NormalizePattern();

Wiederverwendung von Mustern

- Der errechnete Weg wird in 2 Arrays abgespeichert
  - pathRow[i]
  - pathCol[i]
  - Beispiel Rechteckmuster
    - pathRow[0] = 10
    - pathCol[0] = 3
- NormalizePattern();
  - Muster wird in relativen statt absoluten Koordinaten ausgedrückt
  - Subtraktion Wegkoordinaten - Startkoordinaten

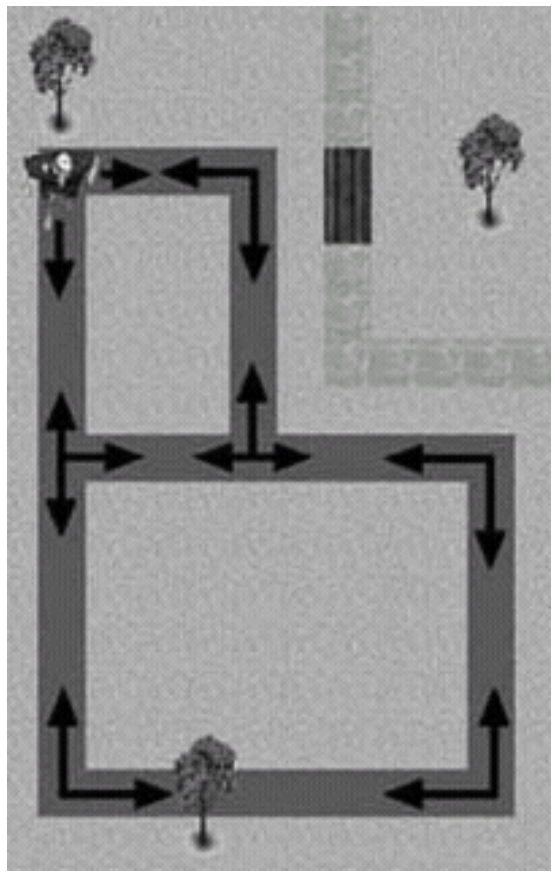
    → Startpunkt (0,0)
  - Muster kann jetzt für andere Startpunkte wiederverwendet werden
    - durch Addition

Bewegungen
- Einflussfaktoren

- Einfache Bewegungsmuster zu statisch und vorhersehbar
- Computer-gesteuerte Figur reagiert auf Spielerbewegungen
  - verlässt das Bewegungsmuster wenn Spieler zu nahe kommt
  - Wird in attacking oder chasing modus versetzt
- Zufallsfaktoren
  - bei komplexeren Mustern
  - Figur kommt an eine Kreuzung
  - Auswahl des Weges zufallsabhängig

Zufallsfaktoren
- Umsetzung

- Erstellen einer pattern matrix
  - 2-dimensionales Array
  - Elemente entweder 0 oder 1
    - Computerfigur darf nur auf Positionen mit 1 (dunkelgrau)
    - Bresenham (modifiziert) setzt die Punkte entlang des Weges =1
- Mögliche Wege
  - possRowPath[], possColPath[]
    - Überprüfung der (8) benachbarten Kacheln
  - Zufällige Auswahl einer Richtung
  - Problem: hin-und-her Bewegungen vermeiden



Implementierung in physikalisch simulierter Umgebung

- Physically Simulated Environments
  - kontinuierlicher Raum
- Methoden des gerasterten Modells unangemessen
  - fertige, starre Muster nutzen Möglichkeiten einer physikalisch simulierten Umgebung kaum aus
- Höhere Intelligenzebene
  - *Let the physics take control of movement*
  - Physics Engine
    - hat letztendlich die Kontrolle über das Objekt
    - führt die Anweisungen aus
    - Control Forces werden an Physics Engine übergeben
    - Resultierendes Verhalten abhängig von allen Inputs

      → z.B. Wenderadius ist bei höherer Geschwindigkeit größer

Pattern Movement
– Umsetzung

- Gerastertes Modell
  - einfache Anweisungen wie moveForward, turnLeft u.s.w. werden schrittweise ausgeführt
- Physikalisch simuliert: andere Herangehensweise
  - Anweisungen werden über einen Zeitraum ausgeführt

    → Game Loop funktioniert nicht als Taktgeber
  - diese Informationen sind in den Pattern Arrays enthalten
  - Übergabe an Physics Engine
    - führt Befehle aus

Pattern Movement
– Codierung

- **Control Data Structure** – Beispiel

  boolean thrusterActive;
  - Aktivierung des Antriebs

  double desiredDistance;
  - wie weit soll sich das Objekt bewegen?

  boolean limitPositionChange;
  - zum Vergleich von zurückgelegter und gewünschter Strecke (Bedingung für Abbruch)

- **Pattern Arrays**
  - jedes Element enthält die gewünschten pattern instructions gemäß der Control Data Structure
  - nächstes Element (Befehlsmenge) wird aufgegriffen wenn Bedingungen erfüllt

    → Ausführung über einen Zeitraum

Pattern Movement
– Funktionsweise des Algorithmus I

- Computer wendet erste Befehlsmenge auf das entsprechende Objekt an
- Physics Engine führt Instruktionen solange aus, bis die Bedingungen in der Befehlsmenge erfüllt sind
- anschließend nächste Befehlsmenge
- arbeitet mit relativen Werten

Pattern Movement

– Funktionsweise des Algorithmus II

- Algorithmus verwendet relative Veränderungen
  - Position
  - Blickrichtung
- Veränderungen müssen nach Ausführung einer Befehlsmenge in die nächste auszuführende Befehlsmenge übertragen werden
- Z.B. Speicherung zweier Vektoren, die die Pos. und Blickrichtung eines Objektes enthalten
- Update jedes Mal wenn eine neue Befehlsmenge geholt wird

Ausführung der Bewegungsmuster

- Beim ersten Starten des Programms
  - Initialisiere Pattern
  - Anfangspositionen und
  - Anfangs-Blickrichtung der Figuren
  - CurrentControlID = 0;
    - Index des aktuellen Elementes im Pattern Array
  - Änderungen auf 0 setzen
    - Position
    - Blickrichtung

Ausführung der Bewegungsmuster

- Funktion *DoPattern* führt all diese Instruktionen aus
  - Übergabe-Parameter
    - Pointer zu Pattern Array
    - Größe des Arrays
  - gibt einen boolean Wert aus
    - True: das Pattern Array ist noch nicht durchgelaufen
    - False: das Pattern Array wurde vollständig durchlaufen
- *DoPattern* wird ständig während der simulation loop aufgerufen
  - um die Steuerungsbefehle in den Mustern auszuführen
  - um das Array zu durchlaufen
  - um das aktuelle Array abzubrechen und ein anderes zu initialisieren
    - Abbruchbedingungen vielfältig, je nach Spiel

DoPattern
- Teilfunktionen

- Überprüfungen
  - gewünschte Position / Blickrichtung erreicht?
    → gehe zu nächster Menge von control instructions
- Berechnungen
  - der Veränderungen der Blickrichtung seit dem Zeitpunkt, als aktuelle Anweisungen initialisiert wurden
  - der Veränderungen der Position seit dem Zeitpunkt, als aktuelle Anweisungen initialisiert wurden
- Bestimmt Faktoren, die die Steuerung / Bewegung des Objektes betreffen
  - Physikalische Simulation
  - Z.B. Minimale Steuerungskraft = 0.05
  - Abbremsen des Objektes nicht abrupt
    - Overshooting

Vergleich

	Gerastert	Phys. simuliert
Inputwerte	absolut	relativ
Ausführung Bewegungen	step-by-step (Game Loop)	best. Zeitraum
Bewegungen	diskret	stetig
Control Data	einfache Anweisungen	Berücksichtigung Bewegungsdauer
Einschränkung	keine	Physikalisches Modell

Quellenangaben

- **Dave C. Pottinger: „Coordinated Unit Movement“**
  Game Developer Magazine, January 1999
  http://www.gamasutra.com/features/game_design/19990122/movement_01.htm

- **Craig W. Reynolds: „Steering Behaviours for Autonomous Characters“**
  Game Developers Conference, 1999
  http://www.red3d.com/cwr/papers/1999/gdc99steer.pdf

- **J.E. Bresenham: „Algorithm for computer control of a digital plotter“**
  http://www.research.ibm.com/journal/sj/041/ibmsjIVRIC.pdf

???

Ich beantworte gerne Ihre Fragen