Game Technology

Optional Lecture 15 – 13.2.2015
Scripting

TECHNISCHE UNIVERSITÄT DARMSTADT

Dr.-Ing. Florian Mehm
Dipl-Inf. Robert Konrad

Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

Preliminary timetable

TECHNISCHE UNIVERSITÄT DARMSTADT

Lecture No.	Date	Topic
1	17.10.2014	Basic Input & Output
2	24.10.2014	Timing & Basic Game Mechanics
3	31.10.2014	Software Rendering 1
4	07.11.2014	Software Rendering 2
5	14.11.2014	Basic Hardware Rendering
6	21.11.2014	Animations
7	28.11.2014	Physically-based Rendering
8	05.12.2014	Physics 1
9	12.12.2014	Physics 2
10	19.12.2014	Procedural Content Generation
11	16.01.2015	Compression & Streaming
12	23.01.2015	Multiplayer
13	30.01.2015	Audio
14	06.02.2015	AI
15	**13.02.2015**	**Scripting - Optional**

Exam

TECHNISCHE UNIVERSITÄT DARMSTADT

The exam is designed for 90 minutes

We are working on the remaining exercises and will announce the bonus points

Be there at 9 (latest), we will start around 9:15

Example exam questions

- For each lecture, we will add a document with the most important points of the lecture and some example questions on the wiki
- No solutions will be provided → questions might also be in the exam

AI Recap: Doom (1993)

TECHNISCHE UNIVERSITÄT DARMSTADT

Source code available at
https://github.com/id-Software/DOOM

Basics

- Characters displayed as sprites with 8 directions

AI Recap: Doom (1993)

TECHNISCHE UNIVERSITÄT DARMSTADT

Movement → Kinematic

Basic moving

- Character is given a move direction and a speed
- If valid (no wall, no falling, …) character is moved
- If bumps into an openable door, game opens it
- Facing the move direction: adjusted in 90 degree steps

Chasing

- Find a chase direction
- Followed for a fixed number of frames (movecount), then re-evaluated
  - Direct path if available
  - If no direct path, try random

AI Recap: Doom (1993)

TECHNISCHE UNIVERSITÄT DARMSTADT

Pathfinding

- Nada
- Not implemented until Quake 3?

Decision Making

- Finite State Machine
- Also combined with the animation sprite handling
- Callback to A_XYZ functions for actions

```
typedef struct
{
  spritenum_tsprite;
  longframe;
  longtics;
  // void(*action) ();
  actionf_taction;
  statenum_tnextstate;
  longmisc1, misc2;
} state_t;
```

```
{SPR_POSS,0,10,{A_Look},S_POSS_STND2,0,0},// S_POSS_STND
{SPR_POSS,1,10,{A_Look},S_POSS_STND,0,0},// S_POSS_STND2
{SPR_POSS,0,4,{A_Chase},S_POSS_RUN2,0,0},// S_POSS_RUN1
{SPR_POSS,0,4,{A_Chase},S_POSS_RUN3,0,0},// S_POSS_RUN2
{SPR_POSS,1,4,{A_Chase},S_POSS_RUN4,0,0},// S_POSS_RUN3
{SPR_POSS,1,4,{A_Chase},S_POSS_RUN5,0,0},// S_POSS_RUN4
{SPR_POSS,2,4,{A_Chase},S_POSS_RUN6,0,0},// S_POSS_ATK2
```

AI Recap: Doom (1993)

TECHNISCHE UNIVERSITÄT DARMSTADT

Strategy

- Nothing here yet

World Interface

- Checking if player is visible
  - Distance, 180 degrees vision, Obstructed by geometry?
- Activated by sounds
  - AI sounds alert them to the player, sound propagation approximated

Execution Management

- „Thinkers“ for game objects (including AI) called once per tic

Content Creation, Scripting

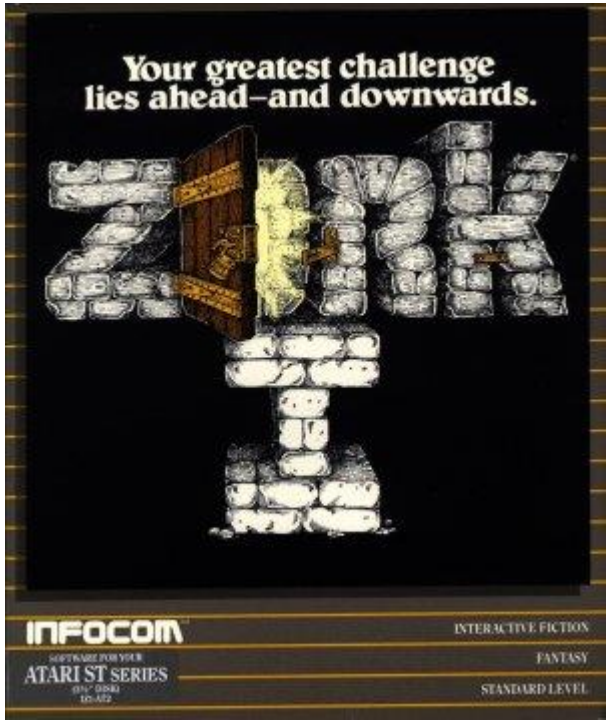
- Scripting implemented in Hexen
- Content Creation: Doomed Editor, add objects with special relevance (triggering events), but event itself hardcoded

History

Zork Implementation Language (ZIL)

- 1979
- Created by Infocom to facilitate the creation of interactive fiction titles
- Compiles to code for a virtual machine → Z-Machine

```
<ROOM LIVING-ROOM
      (LOC ROOMS)
      (DESC "Living Room")
      (EAST TO KITCHEN)
      (WEST TO STRANGE-PASSAGE IF CYCLOPS-FLED ELSE
           "The wooden door is nailed shut.")
      (DOWN PER TRAP-DOOR-EXIT)
      (ACTION LIVING ROOM-F)
      (FLAGS RLANDBIT ONBIT SACREDBIT)
      (GLOBAL STAIRS)
      (THINGS <> NAILS NAILS-PSEUDO)>
```



History

AGI – Adventure Game Interpreter

- 1984
- Created by Sierra On-Line for graphical adventure games
- First used fully in King‘s Quest
- Superseded by SCI – Sierra Creative Interpreter

```
if (said("look","door")) {
  if (posn(ego,0,120,159,167)) {
    print("These doors are strongly
built
      to keep out unwanted visitors.");
  }
  else {
    print("You can't see them from
      here.");
  }
}
```

History

SCUMM – Script Creation Utility for Maniac Mansion

- 1987
- Created by Lucasfilm Games for… Maniac Mansion

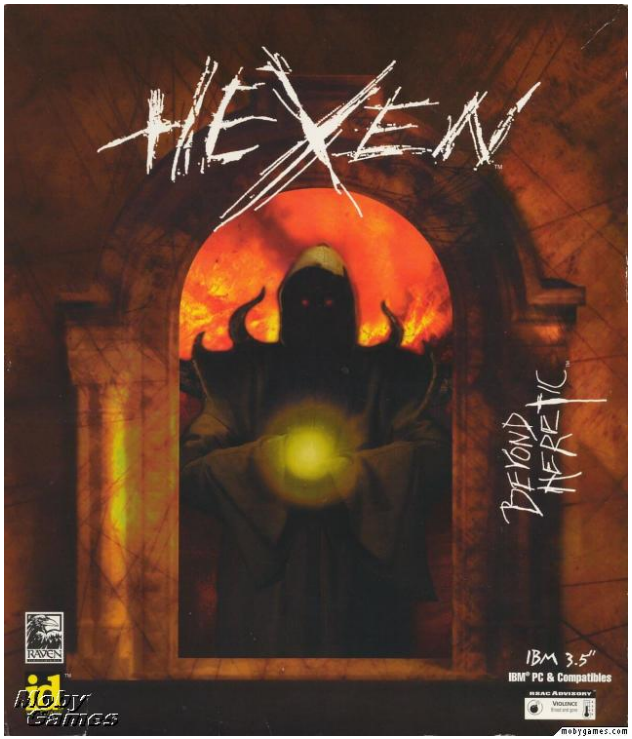
```
cut-scene {
    ...
    actor nurse-edna in-room edna-bedroom
at 60,20
    camera-follow nurse-edna
    actor nurse-edna walk-to 30,20
    wait-for-actor nurse-edna
    say-line nurse-edna "WHATS'S YOUR
POINT ED!!!"
    wait-for-talking nurse-edna
    ...
}
```

History

Action Code Script

- 1995
- Created by Raven Software for Hexen, extending the original Doom engine
- Allowed scripting events during a level

```
SCRIPT 4 (void)
{
  suspend;
  suspend; // The statements "absorb" the
effect of the two first toggles, whichever
they are
  ambientsound("Chat",127);
  printbold(s:"SEQUENCE COMPLETED!");
}
```



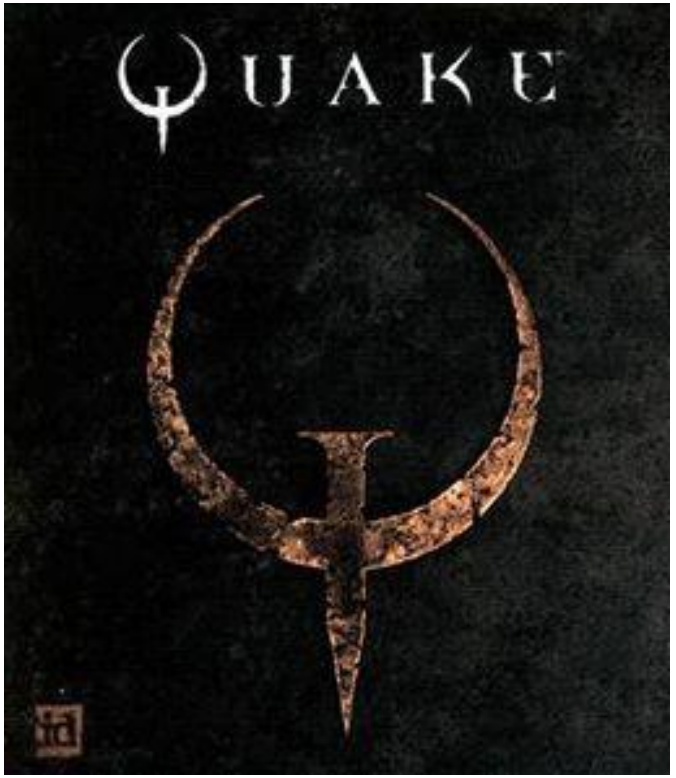
History

TECHNISCHE UNIVERSITÄT DARMSTADT

QuakeC

- 1996
- Created by id Software to control Quake

```
void (float v) ai_berserk =
{
  if (self.health > 25)
  {
    ai_run (v);
  }
  else
  {
    ai_run (v * 1.5); // adjust to your
taste
    self.nextthink = time + 0.075; //
adjust to your taste
  }
};
```



History

TECHNISCHE UNIVERSITÄT DARMSTADT

UnrealScript

- Developed in 1998 for the first Unreal

Can extend the class hierarchy of Unreal

```
class HappyHUD extends HUD
        config(Game);

//VARS
var String joyMessage;
var FONT joyFont;

…
function DrawHUD()
{

     local string StringMessage;

     //projection of ray hit location
  local vector HitLocation, HitNormal;

  //get origin and dir
  Canvas.DeProject(MousePosition, WorldOrigin, WorldDirection);

  StringMessage = "MouseX" @ MousePosition.X @ "MouseY" @ MousePosition.Y;
```

Advantages of Scripting Languages

TECHNISCHE UNIVERSITÄT DARMSTADT

Designer-Friendly

- Designers, non-programmers are enabled to work on the game directly, without needing programmer resources (ideally…)
- Quickly change values, …

Easy to learn

- Often reduced complexity compared to C++ or similar languages
- Often no memory management, pointers, …

Adaptable

- Many scripting languages are flexible
- E.g. Ruby or Lua allow adapting the language itself, e.g. to create a domain-specific language

Advantages of Scripting Languages

TECHNISCHE UNIVERSITÄT DARMSTADT

Concurrency

- Coroutines
- Functions that can be interrupted and continued

No compilation

- No additional time during compiling the game (engine)
- Can be switched during runtime
- Downside: Often slower than compiled code

Mod-support

- Allows players to change the game using the scripting language
- Increases shelf-life

Runtime vs. Data Definition

TECHNISCHE UNIVERSITÄT DARMSTADT

Data-definition languages

- Create data structures that control the game engine
- E.g. LISP-dialect used by Naughty Dog

```
(define-export *player-start*
(new locator
:trans *origin*
:rot (axis-angle->quaternion *y-axis* 45)
))
```

Runtime scripting languages

- Control the game during runtime
- All examples in the history slides are of this kind

Common language properties

TECHNISCHE UNIVERSITÄT DARMSTADT

Interpreted

- Flexibility, portability and rapid iteration
- Virtual machine → port the VM to port the scripts

Lightweight

- Simple, low memory footprints

Support for rapid iteration

- Quicker turnaround time
- See changes immediately/after a restart

Convenience

- Tuned for the purpose in the game

Textual Languages

TECHNISCHE UNIVERSITÄT DARMSTADT

All languages we have seen so far

Special case: Natural-language Programming

Can be found in Inform 7 (interactive fiction tool)

The shower is here. It is fixed in place. "Opposite the mirror is the shower, which is closed." The description of the shower is "When it's open, you get in it to take a shower. Right now it's closed, keeping you from using it."

Instead of opening or entering the shower, say "It is locked down until after the ship makes its jump to hyperspace."

Textual Languages

TECHNISCHE UNIVERSITÄT DARMSTADT

All languages we have seen so far

Special case: Natural-language Programming

Can be found in Inform 7 (interactive fiction tool)

The shower is here. It is fixed in place. "Opposite the mirror is the shower, which is closed." The description of the shower is "When it's open, you get in it to take a shower. Right now it's closed, keeping you from using it."

Instead of opening or entering the shower, say "It is locked down until after the ship makes its jump to hyperspace."

Lua

TECHNISCHE UNIVERSITÄT DARMSTADT

http://www.lua.org/

Development started in 1993 at Pontifical Catholic University of Rio de Janeiro

Small language core

„Events“

- Fired when operators/functions are called, …
- Native code can register to handle them

Tags

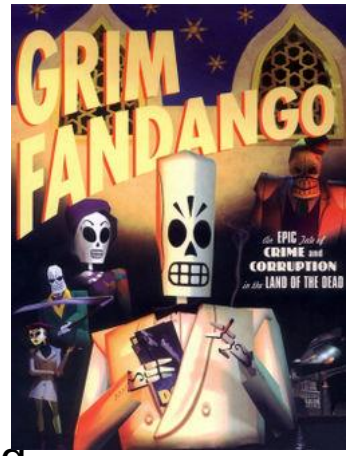
- Code called when events are fired
- Allow Lua behaviour itself to be changed

Lua Example

TECHNISCHE UNIVERSITÄT DARMSTADT

Used in Grim Fandango

http://www.lua.org/wshop05/Mogul.pdf



- – Dialogue
- – Puzzle logic
- – UI/controls
- – Menus
  - Engine handles only animations, backgrounds, sound, rendering, choreography, etc etc etc... But those aren't Grim Fandango

Python

TECHNISCHE UNIVERSITÄT DARMSTADT

https://www.python.org/

Development started in 1989 by Guido van Rossum as a hobby project

Easier to learn for non-programmers than other languages

Disadvantages: Large size and speed

- Relies on hash table lookups

Eve Online server almost completely written in Stackless Python

Visual Languages

TECHNISCHE UNIVERSITÄT DARMSTADT

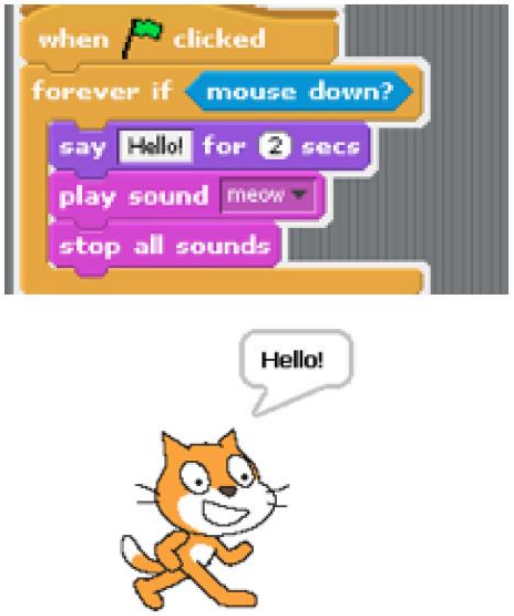
Emerging trend in game tools

Designer-friendly, easy to debug/visualize scripts

Can become complex if the wrong level of abstraction is chosen

Visual Languages: Scratch, Storytelling Alice

TECHNISCHE UNIVERSITÄT DARMSTADT



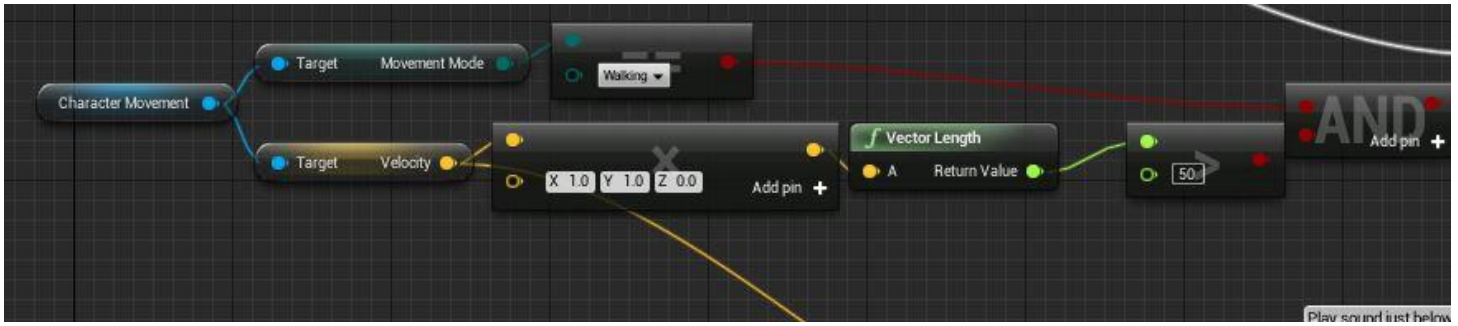
Visual Language: Unreal Blueprint

TECHNISCHE UNIVERSITÄT DARMSTADT

Added in Unreal Engine 4

Can modify almost everything in the game

Graph-based scripting language

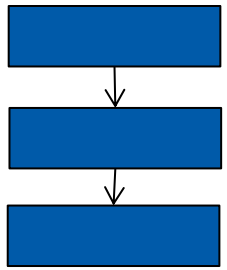


Graph types for visual scripting languages

TECHNISCHE UNIVERSITÄT DARMSTADT

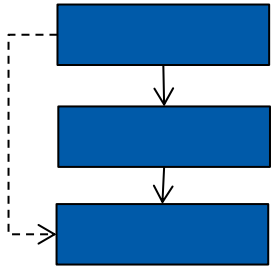
Single branch tree

- Analogue to function without conditions or jumps
- Easiest to implement, but very inflexible



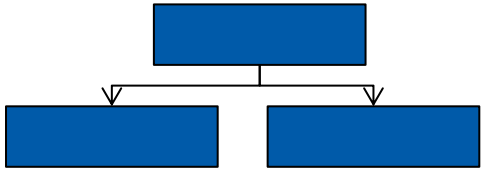
Single branch tree with jumps

- Uses in first version of Unity Adventure Creator
  - …probably out of necessity



Trees

- Allow conditions to be visualized effectively

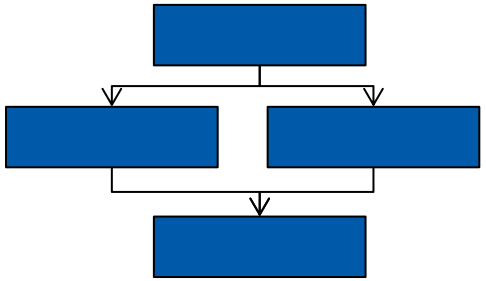


Graph types for visual scripting languages

TECHNISCHE UNIVERSITÄT DARMSTADT

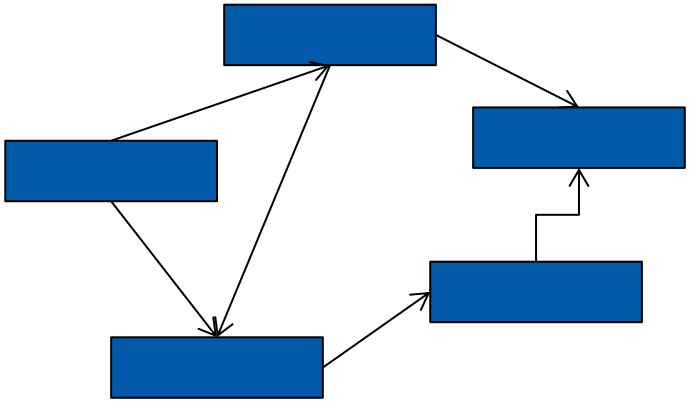
Series-Parallel Digraphs

- Source and Drain similar to electrical circuit
- Always ends in a node
- Advantage over tree: Can be be layouted well automatically, works for scripts with branches and a common end



Graph

- No restrictions
- Most often freely drawn by designer
- Hard to layout automatically
- → Most common for visual scripting languages in games



Data in visual scripting languages

TECHNISCHE UNIVERSITÄT DARMSTADT

Passed around implicitly

- Arguments to individual actions
- Look up, e.g. from a blackboard architecture

Passed around explicitly in the graph

- Exit slots for output variables
- Input slots for input
- Advantage: Can create nodes to change input

Architectures for Scripting

TECHNISCHE UNIVERSITÄT DARMSTADT

Scripted Callbacks

- Most of the behaviour is hard-coded
- Code calls hook functions that are implemented in scripting language

Scripted Event Handlers

- Special case of callbacks
- Allows game objects to react to certain types of events

Extending game object types/define new ones

- Via inheritance or composition/aggregation
- E.g. UnrealScript

Architectures for Scripting

TECHNISCHE UNIVERSITÄT DARMSTADT

Scripted Components or properties

- In component-based game engine architectures
- Define the component by the scripting language
- Used in Dungeon Siege (2002)
- Used by Unity

Script-driven engine systems

- Whole sub-system created in scripting language
- E.g. game object model in script
- Only calls hard-coded parts when needed (e.g. performance-critical parts)

Script-driven games

- Mainly script, game engine more of a library
- E.g. Panda3D

Game Engine Integration

TECHNISCHE UNIVERSITÄT DARMSTADT

Embed the virtual machine (often written in C or C++)

Interface to/from native code

- Functional language
  - Look up the function‘s byte code and run it, providing arguments
- Object-oriented language
  - Create/destroy instances, call member functions
- Two-way communication
  - Allow script functions to call native code
  - Often realized by registering functions with the scripting language
  - Can be automized if the native language supports RTTI (e.g. see Lua integration into C#)

Game Engine Integration

TECHNISCHE UNIVERSITÄT DARMSTADT

Referring to Game Objects

Numerical Handles

- Simple to use/set up
- Can be confusing

Strings with names

- Easier to use
- More memory used, string comparisons, miss-types names

Hashed string ids

- Reduce to integer for the engine

Scripting Finite State Machines

TECHNISCHE UNIVERSITÄT DARMSTADT

As seen in the AI lecture, FSM are often at the core of AI and game logic code

Specific support in the scripting language

- Custom syntax for states
- Mirrored in the game object model
- Example Uncharted Engine
  - Each script can have multiple states
  - Different event handlers, …

Multithreaded scripts

TECHNISCHE UNIVERSITÄT DARMSTADT

Usually done via cooperative multitasking

Scripts explicitly yield to other scripts

- Wait for x seconds
- Wait for x frames

Examples

- Can be realized in Lua
- Unity

Synchronized via signals

- E.g. WAIT_UNTIL(signal)

Exercise 14 – Practical Part

TECHNISCHE UNIVERSITÄT DARMSTADT

1.2

- neighbourhoodSize
  - Boids only look for other boids inside this radius – others are ignored
- neighbourhoodMinDP
  - DP = dot product
  - Boids have a cone in which they can look
  - The size of the cone depends on the minimal dot product between the boid‘s direction and the compared boid

1.3

- The state machine architecture is pretty complex
- → Adaptable to different needs, but needs a lot of glue code

Exercise 14 – Theoretical Part – 2.1

TECHNISCHE UNIVERSITÄT DARMSTADT

Steering behavior	Steering output (vec2)
Steer (kinematic)	Vector to target with max speed Normalize(3, -4) * 2
Flee (kinematic)	Vector away from target with max speed Normalize(3, -4) * -2
Steer (dynamic)	Vector to target with max accel. Normalize(3, -4) * 0.5
Flee (dynamic)	Vector away from target with max speed Normalize(3, -4) * -0.5
Pursue	How long do we take? Length of the way: Length of (3, -4) = 5 How long do we take? 5 / 2 = 2.5 New Position of B: (8, 2) + (3, 4) * 2.5 = (8,2) + (7.5, 10) = (15.5, 12) Steering output: Normalize((15.5, 12) – (5, 6)) * 0.5
Evade	How long do we take? Length of the way: Length of (3, -4) = 5 How long do we take? 5 / 2 = 2.5 New Position of B: (8, 2) + (3, 4) * 2.5 = (8,2) + (7.5, 10) = (15.5, 12) Steering output: Normalize((15.5, 12) – (5, 6)) * -0.5

Exercise 14 – Theoretical Part – 2.2

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Can A* be interrupted during its execution and continued at a later time? If yes, describe why. If not, give a counterexample.

- Yes. If we put the data (open-node-list, …) on the heap, we can continue at another time.

b) Imagine that an interruptible pathfinding algorithm is interrupted and continues 1 second later. Are there problems to be expected in this case? If so, what could the problems be?

- The path could be invalid (terrain or buildings changed/destroyed), the move command could be canceled by the player, weights can have changed, …

c) If a) is true, does this make A* an Anytime algorithm? (i.e. Would the implementation be able to provide a valid path after being interrupted?)

- Not without changes. If the algorithm is interrupted before it finished, it does not have a valid path to the target. You can search for “Anytime A*” to find algorithms with such changes.

Exercise 14 – Theoretical Part – 2.3

TECHNISCHE UNIVERSITÄT DARMSTADT

Imagine a scene with many characters walking in random directions. In this scene, an AI LOD system is used. Nearby characters use collision avoidance, far away characters may freely interpenetrate.

What problems could arise if characters exploded when they collided?

If an AI stays in one of the LOD regions, everything is fine

If AI's interpenetrate (= far LOD) and the player comes closer, they can suddenly explode as they get into the near LOD

Depending on the implementation, characters on the far-side could wander into those close to the border

„Exercise 15“

TECHNISCHE UNIVERSITÄT DARMSTADT

The exercise sheet will contain example projects you can undertake to increase your knowledge and skill with game technologies

We will not review or grade work you did for „Exercise 15“

- but if you want to show us a great game you made, feel free to send us a link ;-)

Thank you!

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Questions & Contact

game-technology@kom.tu-darmstadt.de