the gamedesigninitiative at cornell university

Lecture 14

Data-Driven Design

Take-Away for Today

- What is “data-driven” design?
  - How do the programmers use it?
  - How to the designers/artists/musicians use it?
- What are the benefits of data-driven design?
  - To both the developer and the player
- What is a level editor and how does it work?
  - What can you do graphically?
  - How does scripting work in a level editor?

the gamedesigninitiative at cornell university

Recall: Game Components

- **Game Engine**
  - Software, created primarily by programmers
- **Rules and Mechanics**
  - Created by the designers, with programmer input
- **User Interface**
  - Coordinated with programmer/artist/HCI specialist
- **Content and Challenges**
  - Created primarily by designers

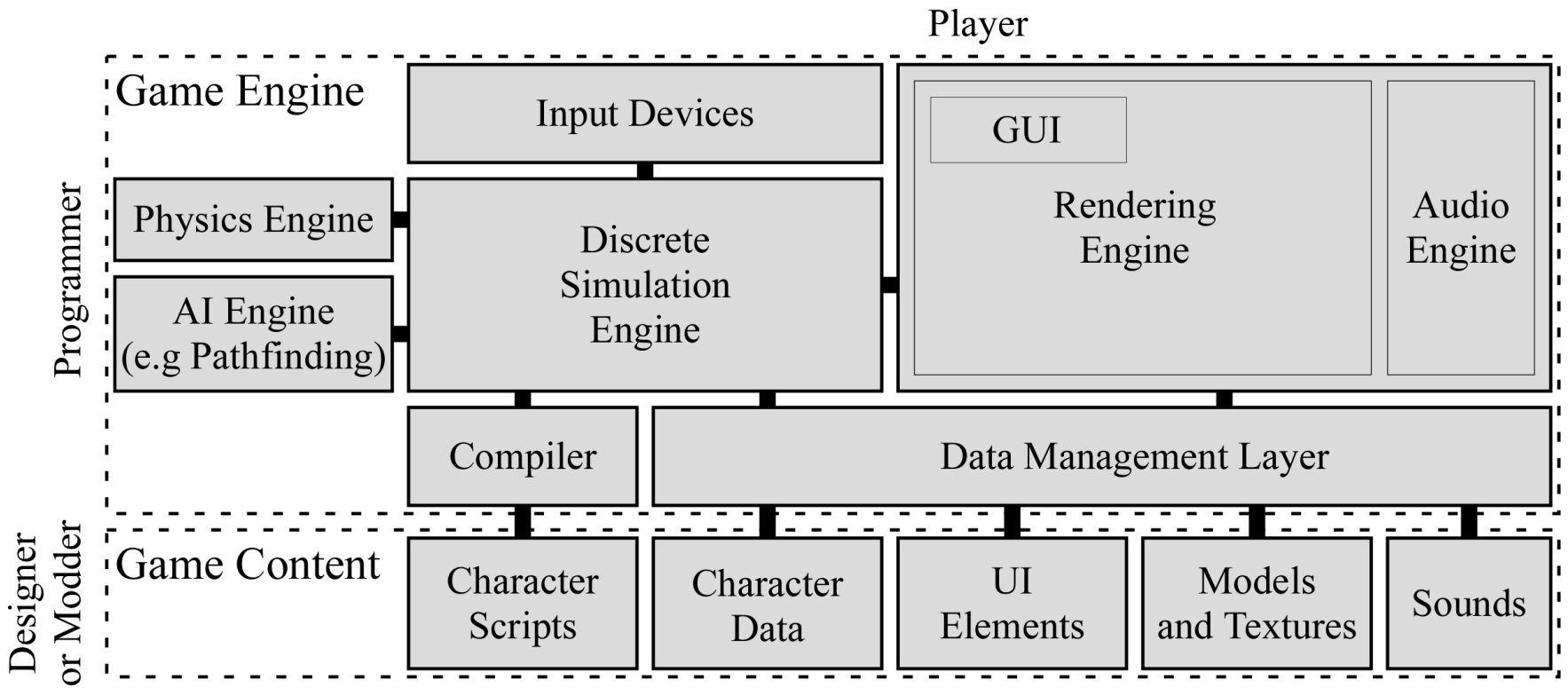
the gamedesigninitiative at cornell university

Data Driven Design

- **No code outside engine**
  - Engine determines space of possibilities
  - Actual possibilities are data/scripts
- **Examples**:
  - Art and music in industry-standard file formats
  - Object data in XML or other data file formats
  - User interface in XML or other data files
  - Character behavior specified through scripts

the gamedesigninitiative at cornell university

Architecture: The Big Picture



the gamedesigninitiative at cornell university

Why Data Driven Design?

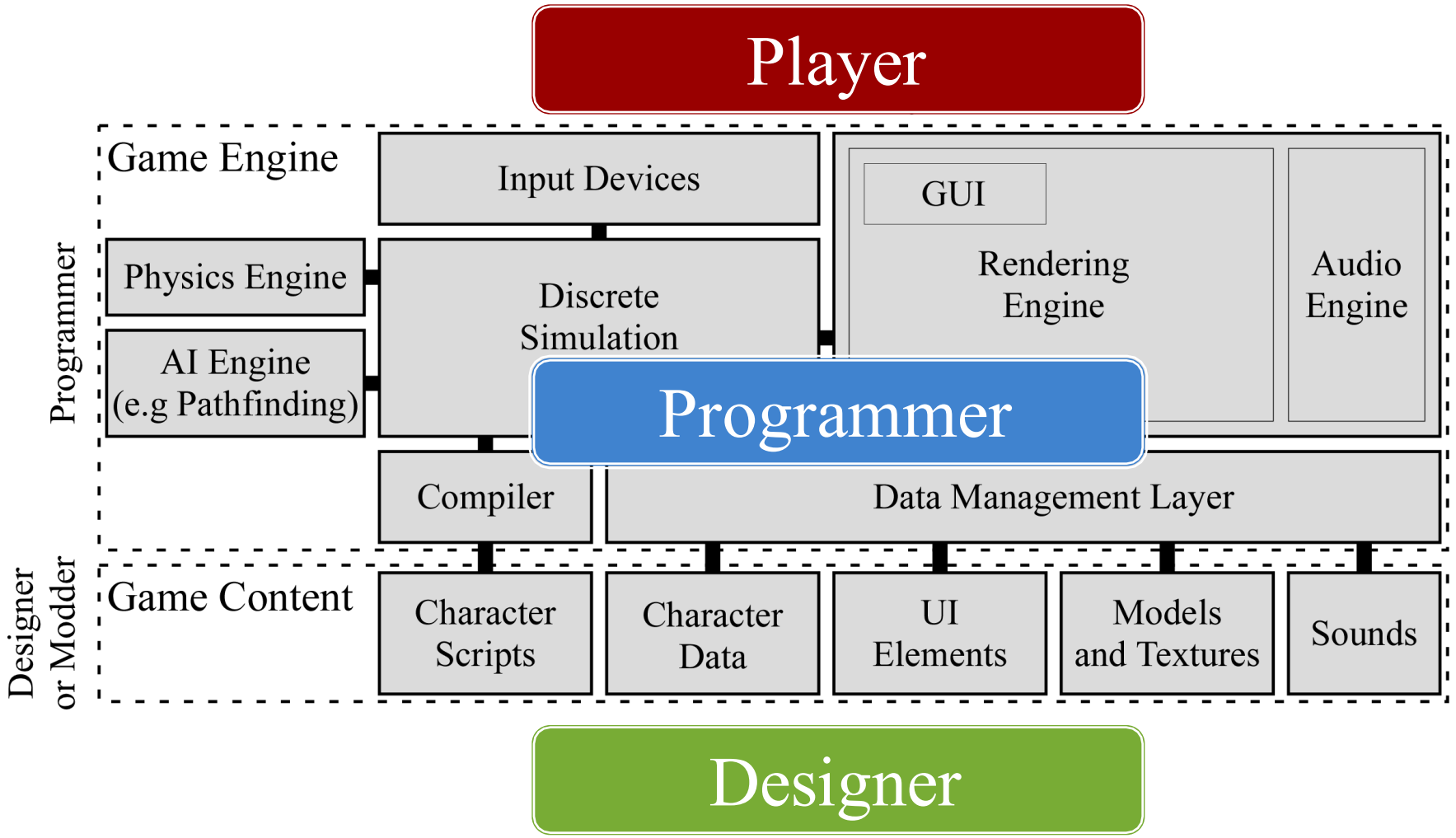
- Games involve many actors:
  - Programmers: Create the game engine
    - Focus on technological development
  - Designers: Create the game content
    - Typically artistic/behavioral content

.....

  - Players: Interact with the game
  - Modders: Modify the game content
    - Post-market "designers"
- Optimize the **production pipeline**

the gamedesigninitiative at cornell university

Architecture: The Big Picture



The Benefits of Modding

- Can extend the life span of the game
  - Keep the game content fresh over many years
  - If gamers are playing, will buy DLC!
- Community can add new game play
  - *Counter Strike* was a community mod
  - New quests and items for *Skyrim*
- Open up game to new markets
  - *Starcraft* in training and education

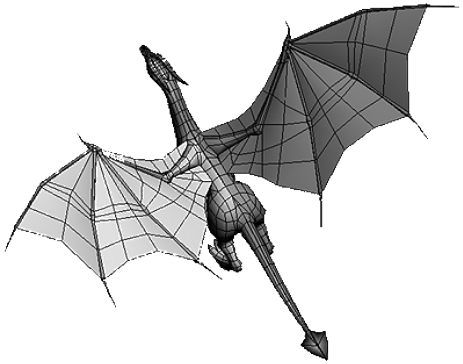
the gamedesigninitiative at cornell university

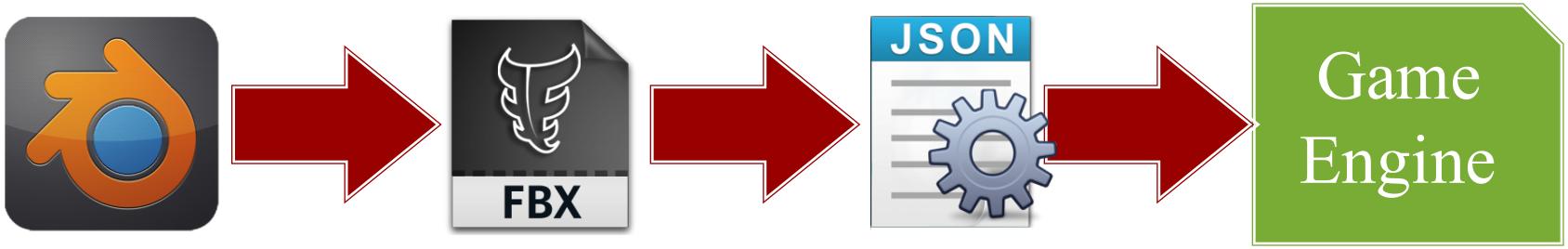
Common Development Cycle

- Start with small number of programmers
- Programmers create a content pipeline
  - Productivity tools for artists and designers
  - Data can be imported, viewed and playtested
- Hire to increase number of artists, designers
  - **Focus**: creating content for the game
- Ship title and repeat (e.g. cut back on artists)

the gamedesigninitiative at cornell university

Content Pipeline

Art Tools	Initial File Format	Final File Format	Software
	AUTODESK **FBX**	**G3DJ**	libGDX



Content Creation Tools

- **Level editor**
  - Create challenges and obstacles
  - Place objects in the world
  - Tune parameters (physics, difficulty, etc.)
- **Scripting Languages**
  - Define character behavior
  - Script triggers and events
  - Layout the user interface

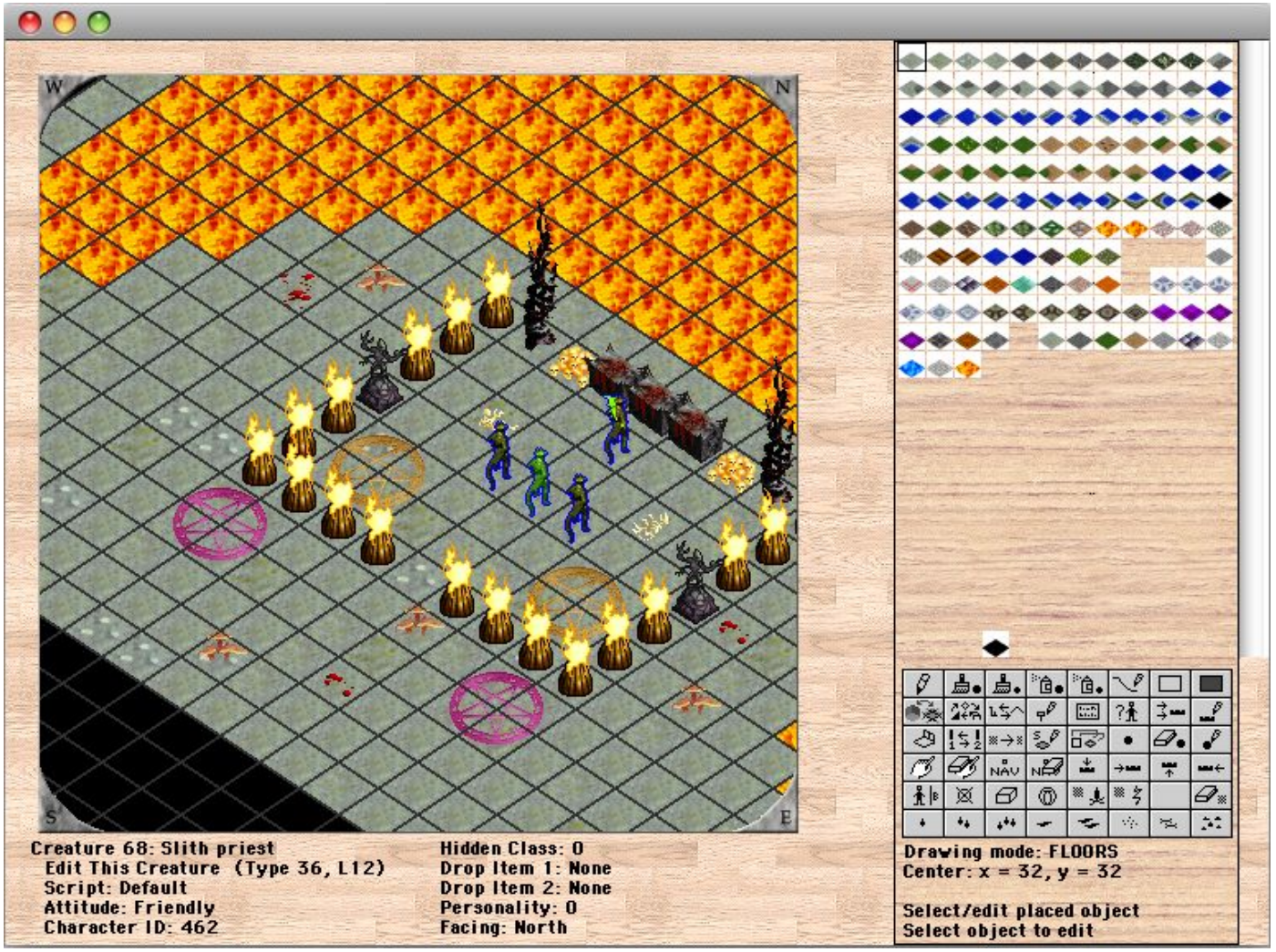
the gamedesigninitiative at cornell university

Level Editor Features

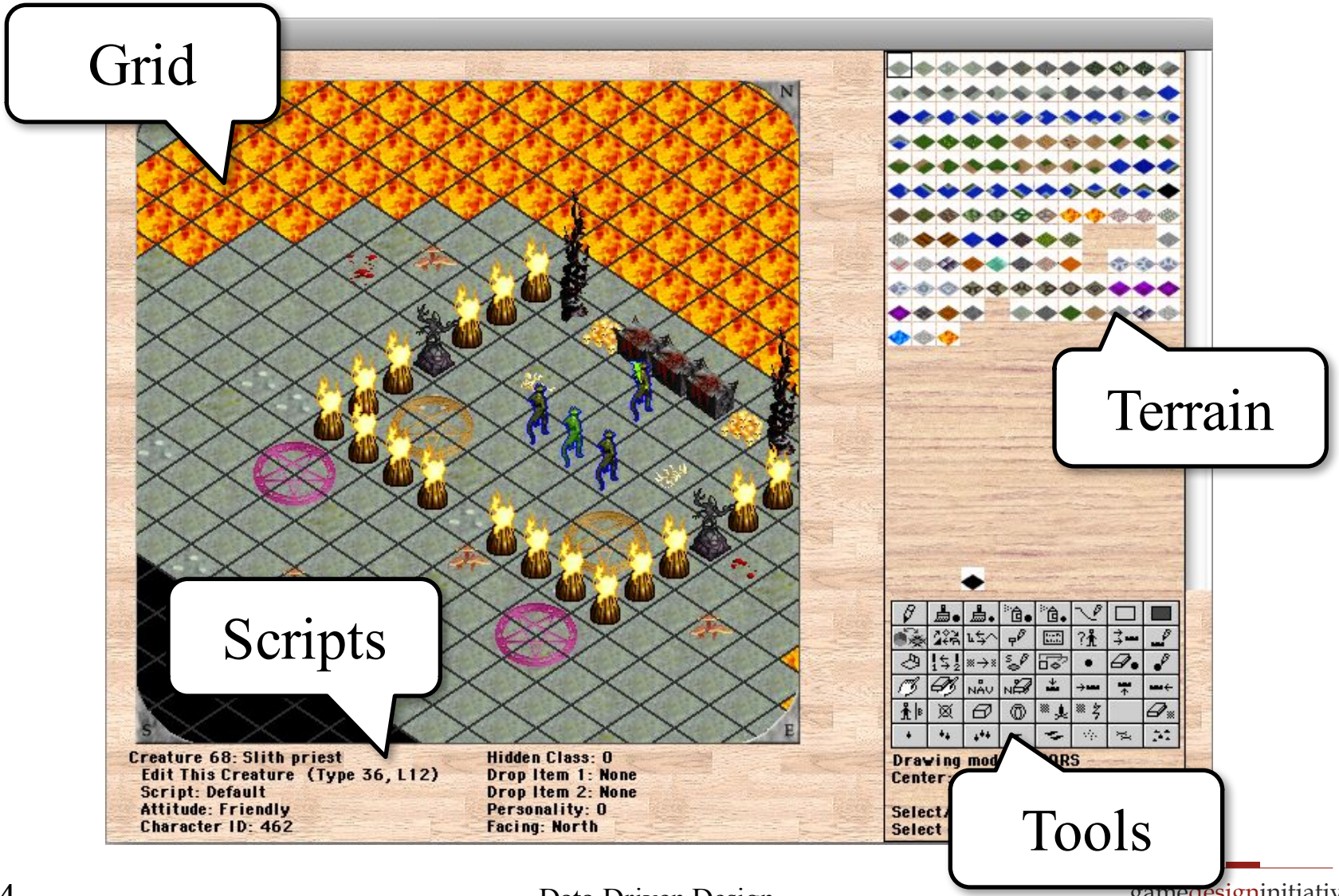
- **Create Terrain**
  - Defines game geometry as 2D or 3D space
  - Terrain can be **free-form** or as **grid tiles**
- **Place Objects**
  - Includes NPCs, hazards, power-ups, etc.
  - Again can be free-form or aligned to a grid
- **Annotate Objects/Terrain**
  - Attach scripts to interactive objects
  - Define boundaries for event triggers

the gamedesigninitiative at cornell university

Example: *Blades of Avernum*



Example: *Blades of Avernum*



gamedesigninitiative at cornell university

Level Editor: **Code Sharing**

- **Option**: level editor in **same project**
  - Single Eclipse project for both
  - **Pro**: Easy to integrate into the game itself
  - **Con**: Harder to separate modules/subsystems
- **Option**: develop **core technology**
  - Identify source code used by each
  - JAR for both level editor and game
  - **Pro**: Cleaner separation of subsystems
  - Con: Harder to iterate the design

the gamedesigninitiative at cornell university

Level Editor: **Serialization**

Stores:
Game Model

Level
File

Level Editor: Serialization

- Do not **duplicate** data
  - Art and music are separate files
  - Just reference by the file name

- Must **version** your file
  - As game changes, format may change
  - Version identifies the current file format
  - Want a conversion utility between versions
  - Version should be part of **file header**

the gamedesigninitiative at cornell university

Levels and Game Architecture

- Game data is **not compiled** into software
  - Files go into a well-define folder
  - Game loads everything in folder at start-up
  - Adding new files to folder adds levels
- But this requires **robustness**
  - What if the levels are missing?
  - What if the levels are corrupted?
  - What if you are using wrong file version?

the gamedesigninitiative at cornell university

Levels and Error Detection

- **Corruption** a major problem in this design
  - Player might trash a level file (or directory)
  - Modder might alter level improperly
  - Content patch might have failed
- Process all errors **gracefully**
  - Check everything at load time
  - If level corrupt, allow play in others
  - Give helpful error messages

the gamedesigninitiative at cornell university

Serialization and XML

Advantages

- **Human readable**
  - Easy for modders
- **Extendible**
  - Easy to add new tags
  - Easy to track versions
- **Portable**
  - Readers on all OSs

Disadvantages

- **Overhead**
  - Parsers not efficient
- **Verbose**
  - Everything is text
  - Tags take up space
- **No Free-Lunch**
  - Only portable if code is

the gamedesigninitiative at cornell university

Modern Alternative: JSON

XML

```
<NPC>
   <type>Orc</type>
   <health>200</health>
   <position>
      <x>50</x>
      <y>25</y>
    </position>
</NPC>
```

JSON

```
{
   "NPC" : {
      "type" : "Orc",
      "health" : 200,
      "position" : {
         "x" : 50,
         "y" : 25
}}}
```


the gamedesigninitiative at cornell university

Modern Alternative: JSON

XML

```
<NPC>
   <type>Orc</type>
   <health>200</health>
   <po
      <x
      <y>25</y>
   </position>
</NPC>
```

XmlReader

JSON

```
{
   "NPC" : {
      "type" : "Orc",
      "
      "p
         "x" : 50,
         "y" : 25
}}}
```

JsonReader

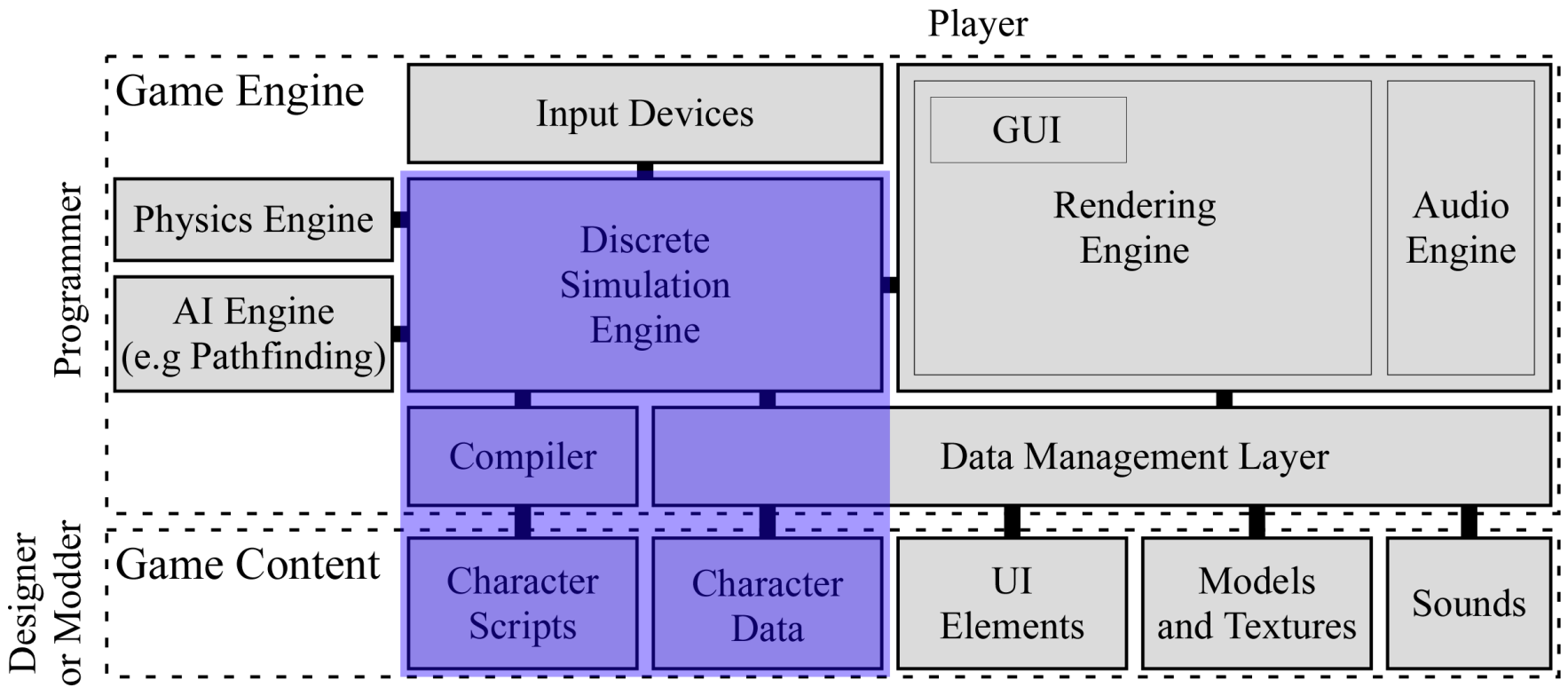
the gamedesigninitiative at cornell university

Content Creation Tools

- **Level editor**
  - Create challenges and obstacles
  - Place objects in the world
  - Tune parameters (physics, difficulty, etc.)
- **Scripting Languages**
  - Define character behavior
  - Script triggers and events
  - Layout the user interface

the gamedesigninitiative at cornell university

Scripting



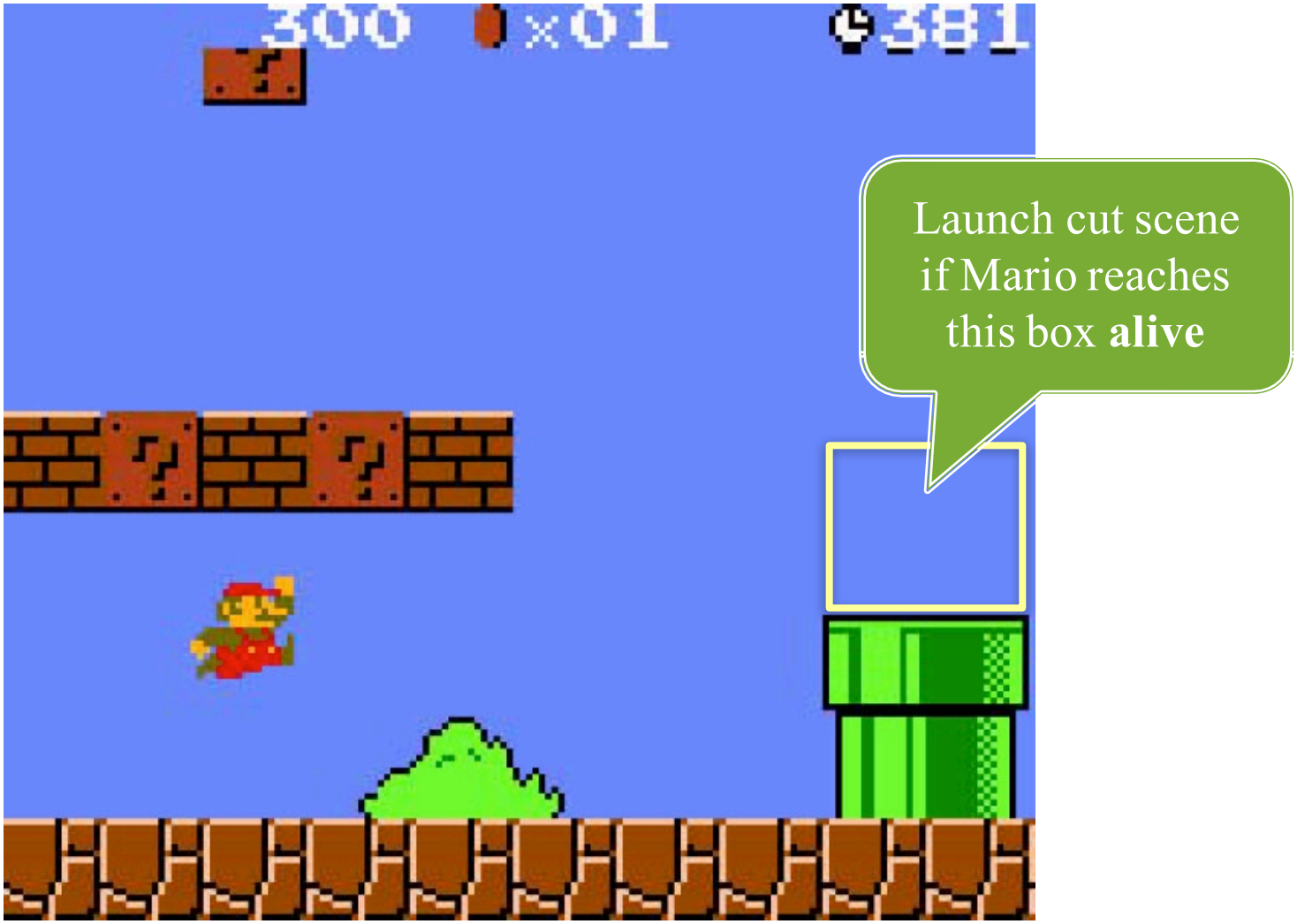
the gamedesigninitiative at cornell university

Why Scripting?

- **Character AI**
  - Software only aware of high level actions
  - Specific version of each action is in a script
- **Triggers**
  - Actions happen in response to certain events
  - Think of as an `if-then` statement
    - **`if`**: check if trigger should fire
    - **`then`**: what to do if trigger fires

the gamedesigninitiative at cornell university

Triggers and Spatial Boundaries



Ways of Scripting

- Static **functions/constants** exposed in editor
  - Script is just the name of function to call
  - Used in the sample level editor
  - Typically good enough for this course
- Use standard **scripting language**
  - **Examples**: Lua, stackless python
  - A lot of overhead for this class
  - Only if writing high performance in C/C++

the gamedesigninitiative at cornell university

Scripting in *Dawn of War 2*

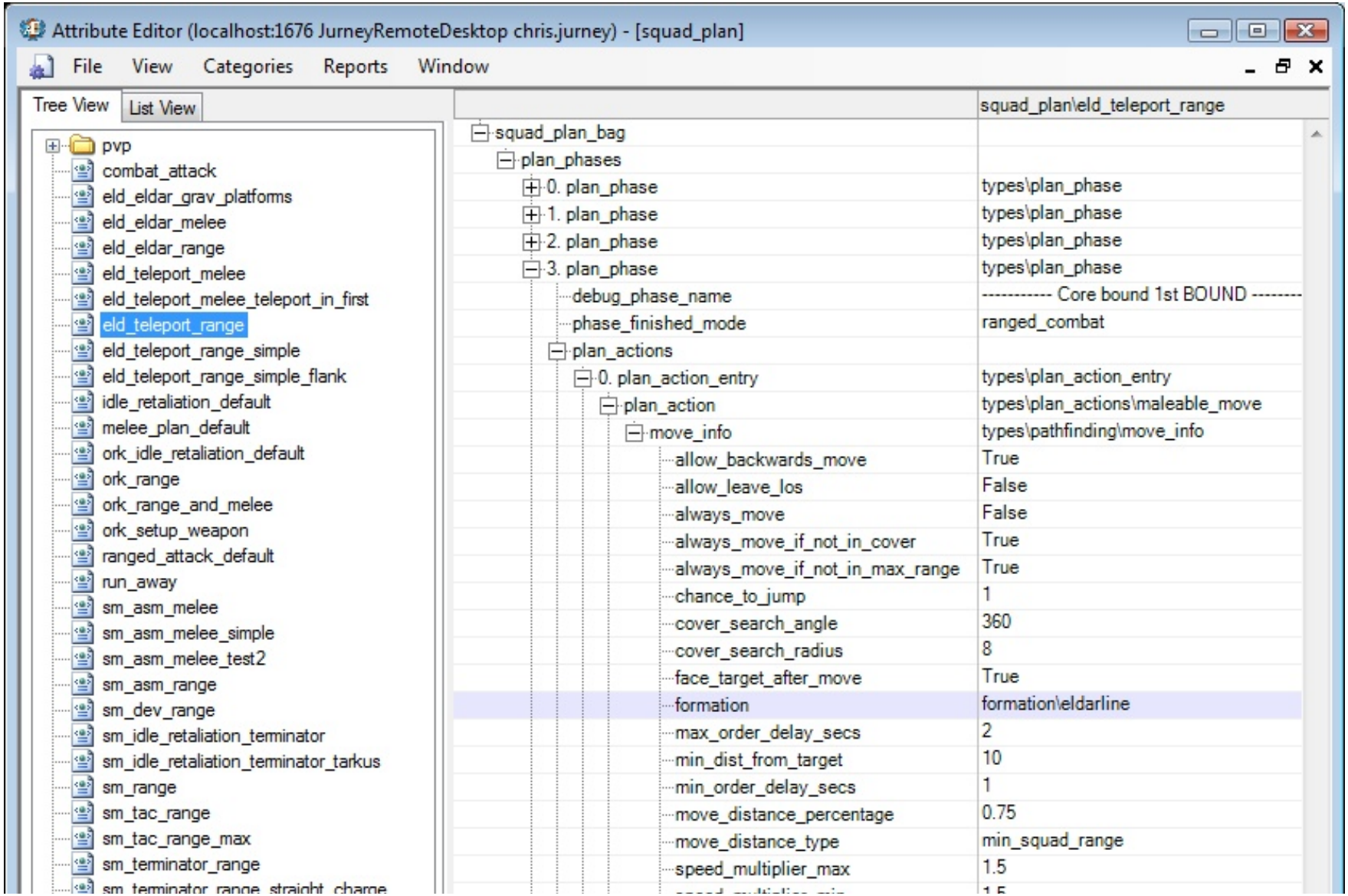
infantry-plan.squadai * Sc1

File Edit Search View Tools Options Language Buffers Help

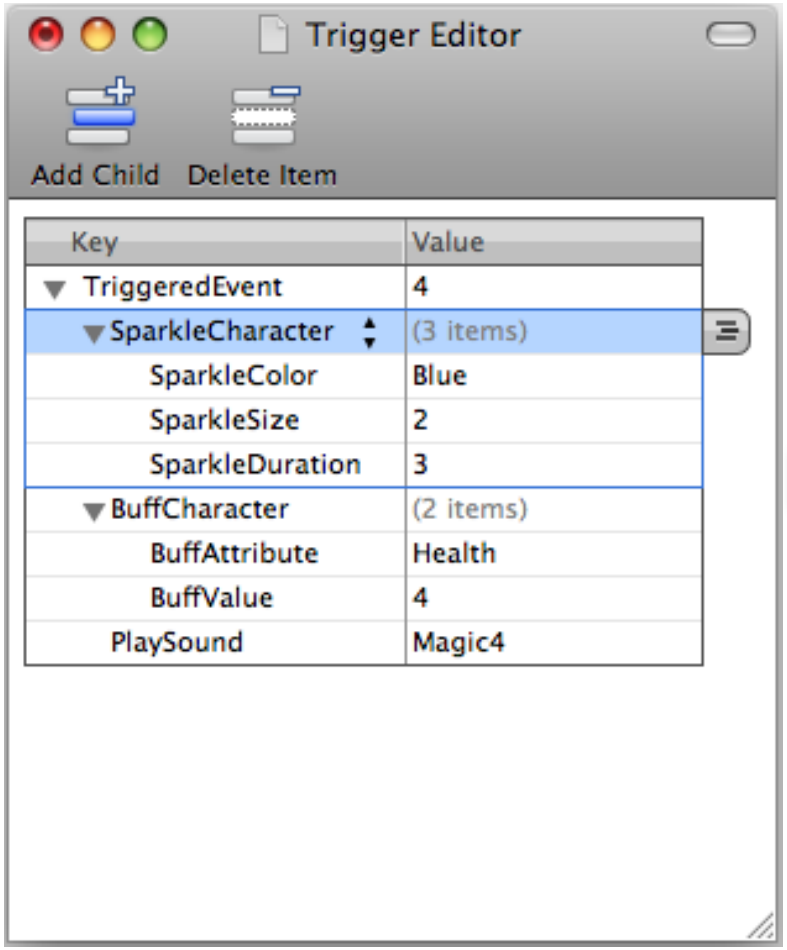
1 assault-building-plan.lua | 2 sniper-plan.squadai | 3 infantry-plan.squadai *

```
16  ------------------------------------------------------------------------------
17  -- plan
18
19  plan =
20  {
21      ------------------------------------------------------------------------------
22      -- phase0: Before First Bound
23          {
24          type = DATA_PHASE,
25          name = "START PLAN: all move NO COVER NO BACKWARDS",
26          --
27          {
28              apply_to = {ET_Core, ET_RFlank, ET_LFlank},
29              actions =
30              {
31                  ACTION_MOVE_POSTURE_EXT( DT_MAX_SQUAD_RANGE, .95, 4.0, 30.0, "squad_formations/squad_ai.lua"
32              }
33          },
34      },
35      {
36          type = DATA_PHASE,
37          name = "1st SQUAD BOUND -- LOOK FOR COVER",
38          --
39          {
40              apply_to = {ET_Core, ET_RFlank, ET_LFlank},
41              actions =
42              {
43                  ACTION_MOVE_POSTURE( DT_MAX_SQUAD_RANGE, 0.85, 10.0, 60.0, "squad_formations/squad_ai.lua",
44              }
45          },
46      },
47
48      ------------------------------------------------------------------------------
49      -- phase2 -- BOUND 1 CORE (core runs in an drops to prone)
50      {
51          type = DATA_PHASE,
52          name = "Core 1st Bound"
```

Simpler: XML Specification



XML as “Scripting Language”

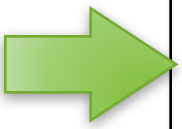



```
<TriggeredEvent id="4">
  <SparkleCharacter>
    <SparkleColor>Blue</SparkleColor>
    <SparkleSize>2</SparkleSize>
    <SparkleDuration>3
     </SparkleDuration>
  </SparkleCharacter>
  <BuffCharacter>
    <BuffAttribute>Health
     </BuffAttribute>
    <BuffValue>4</BuffValue>
    <PlaySound>Magic4</PlaySound>
  </BuffCharacter>
</TriggeredEvent>
```


the gamedesigninitiative at cornell university

XML as “Scripting Language”

```
<TriggeredEvent id=”4”>
  <SparkleCharacter>
    <SparkleColor>Blue</SparkleColor>
    <SparkleSize>2</SparkleSize>
    <SparkleDuration>3
     </SparkleDuration>
  </SparkleCharacter>
  <BuffCharacter>
    <BuffAttribute>Health
     </BuffAttribute>
    <BuffValue>4</BuffValue>
    <PlaySound>Magic4</PlaySound>
  </BuffCharacter>
</TriggeredEvent>
```



```
switch (triggerIdentifier) {
 …
 case 4:
  sparkleCharacter(BLUE,2,3);
  buffCharacter(HEALTH,4);
  playSound(MAGIC4);
  break;
 …
}
```

This is text, not compiled code

Dynamic Code Compilation

- **Reflection** is a form of data-driven design
  - Allows languages to treat code as data
  - In most modern languages (Java, C#, etc.)
- **Dynamic code compilation** aids reflection
  - Read source code from text file as string
  - Generates new object implementing code
- **Primary class**: javax.tools.JavaCompiler

the gamedesigninitiative at cornell university

Automatic Code Generation

- **Goal**: Generate simple code
- **Problem**: Need Java code
  - Reflection creates an object
  - May need subclassing
  - May need memory allocation
- **Solution**: Code Generation
  - Scripter writes minimal code
  - Game engine pads string
  - Compiles result to an object

```
println("Hello!");
```

```
public class HelloClass {
   public static void main(String a[]) {
      System.out.println("Hello!");
   }
}
```


the gamedesigninitiative at cornell university

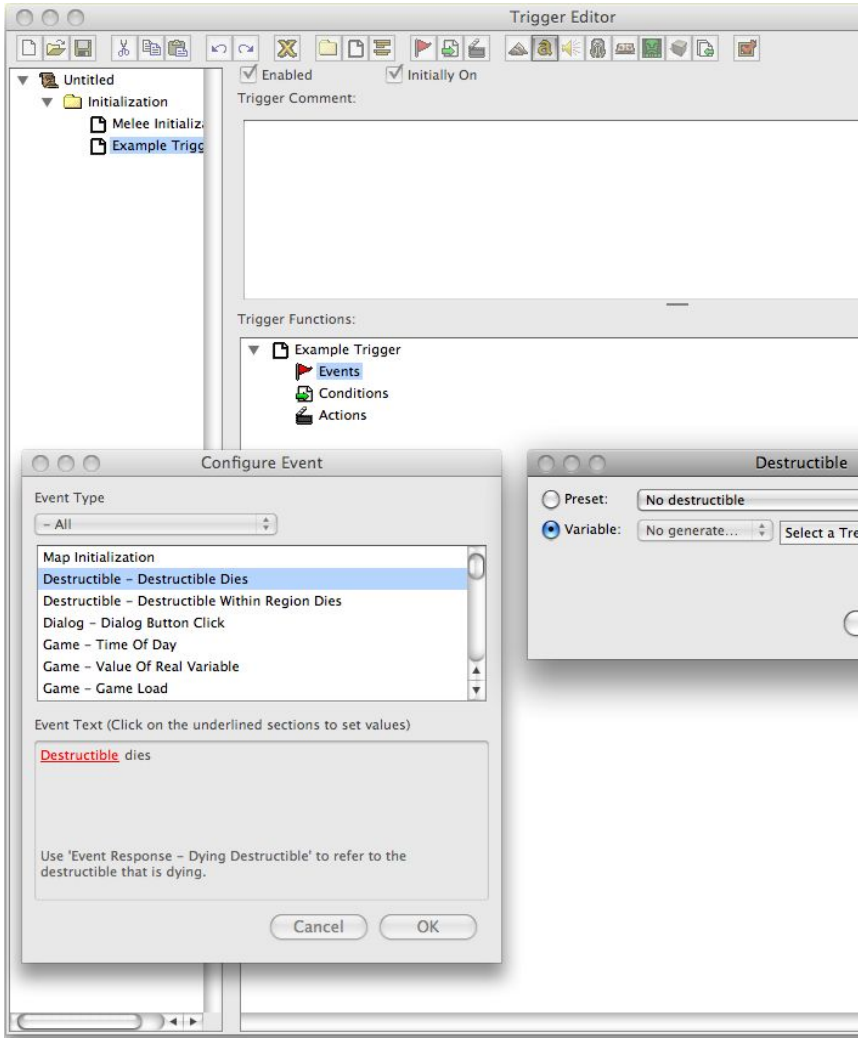
Script Design

- Main if-then statements
- Loops used **sparingly**
  - Recursion = bad!
  - Only **foreach** loops
- Heavy use of function calls
  - Functions in engine
  - Define the **game system**
- No memory allocation

```
// if target in range attack it
if (target_ok() &&
 dist_to_char(get_target()) > 16) {
 set_foe_target(ME,-1);
}

if (get_foe_target(ME,8,0)) {
 do_attack();
}

if (who_shot_me() >= 0) {

set_foe_target(ME,who_shot_me());
 do_attack();
}

if ((get_attitude(ME) >= 10) &&
   (dist_to_pc() <= hunt_range)) {
 set_foe_target(ME,pc_num());
}

if (am_i_doing_action() == FALSE) {
 end_combat_turn();
}
```

If REALLY not Programmers

- Need a scripting GUI
  - Everything from menu
  - Other visualization
    - FSM Graphs
    - iTunes-style rules
- Great for modders
  - Highly restricted scripts
  - Will not crash program



Modders and Scripting

- Giving modders lots of power is **dangerous**
  - Can create malicious plug-ins
    - Show one functionality, but cause another
    - Or may be simply buggy
  - Can create bots or cheats
- Supporting modders is a **serious decision**
  - Can be a tech support nightmare
  - Do or do not; there is no try

the gamedesigninitiative at cornell university

Summary

- Data-driven design has several advantages
  - Faster content production; code reuse is easier
  - Embrace of modder community can add value
- But it comes with some pitfalls
  - Must expect data to become lost/corrupted
  - Modders can introduce malicious data
- Two major focuses in data-driven design
  - **Level editors** place content and challenges
  - **Scripts** specify code-like behavior outside of code

the gamedesigninitiative at cornell university