SCRIPTING CREATING A SIMPLE DOMAIN SPECIFIC LANGUAGE FOR GAM 150 CLUB

Randy Gaul

Overview

- Writing C code takes a long time
- Some games can make use of
  - Lots of one off logic
  - High amounts of scripted sequences
  - Simple AI
- Wouldn’t it be nice to have a tool to make all these things?

Domain Specific Language (DSL)

- A Domain Specific Language is:
  - A language dedicated to a particular problem domain.
- Can be a highly specialized form of a scripting language
- Instead of writing C code
  - Write simple text files
  - Read text files in during runtime
  - No compilation required
  - Hotswapping

StarCraft Brood War

- This game had a “trigger editor”
- Trigger:
  - Collection of conditions and actions
- Condition:
  - Some test to perform, returns boolean
- Action:
  - Some function pointer to call
  - Could be anything
    - Create enemies, update AI, display messages, etc.

Stealing BW's Triggers Design

- Lets call these "triggers" events
- An event has conditions and actions
- If all conditions are met
  - Run all actions
  - Discard the event
- Events can be constructed from files
- Can write events in a text file

Events Example

- Consider a list of events read from file:
  - Event1
    - Conditions – Player is within location “HealingArea”; “HealEnabled” is true
    - Actions – Heal player, decrement “HealCounter”; Set “HealEnabled” to false
  - Event2
    - Conditions – Player is not within location “HealingArea”; “HealEnabled” is false
    - Actions – Set “HealEnabled” to true
  - Event3
    - Conditions – “HealCounter” is at least 3
    - Actions – Destroy all “HealingPools” at “HealingArea”

Events Example – In English

- We have a healing pool
- Player enter pool, gets healed
  - Will not be healed until leaves then returns
- Pool will heal 3 times
  - On the third the pool is destroyed

Scripting Languages

- Scripting in a text file is powerful
- Purpose of scripting is to save time
- You want a scripting language
  - There is a drawback:
    - Must be able to create the language quickly
- If language development:
  - Takes too long
  - Too hard to add new events
  - Then purpose is self-defeating

Implementation

- Need very simple way to parse events from file
- First up: file format

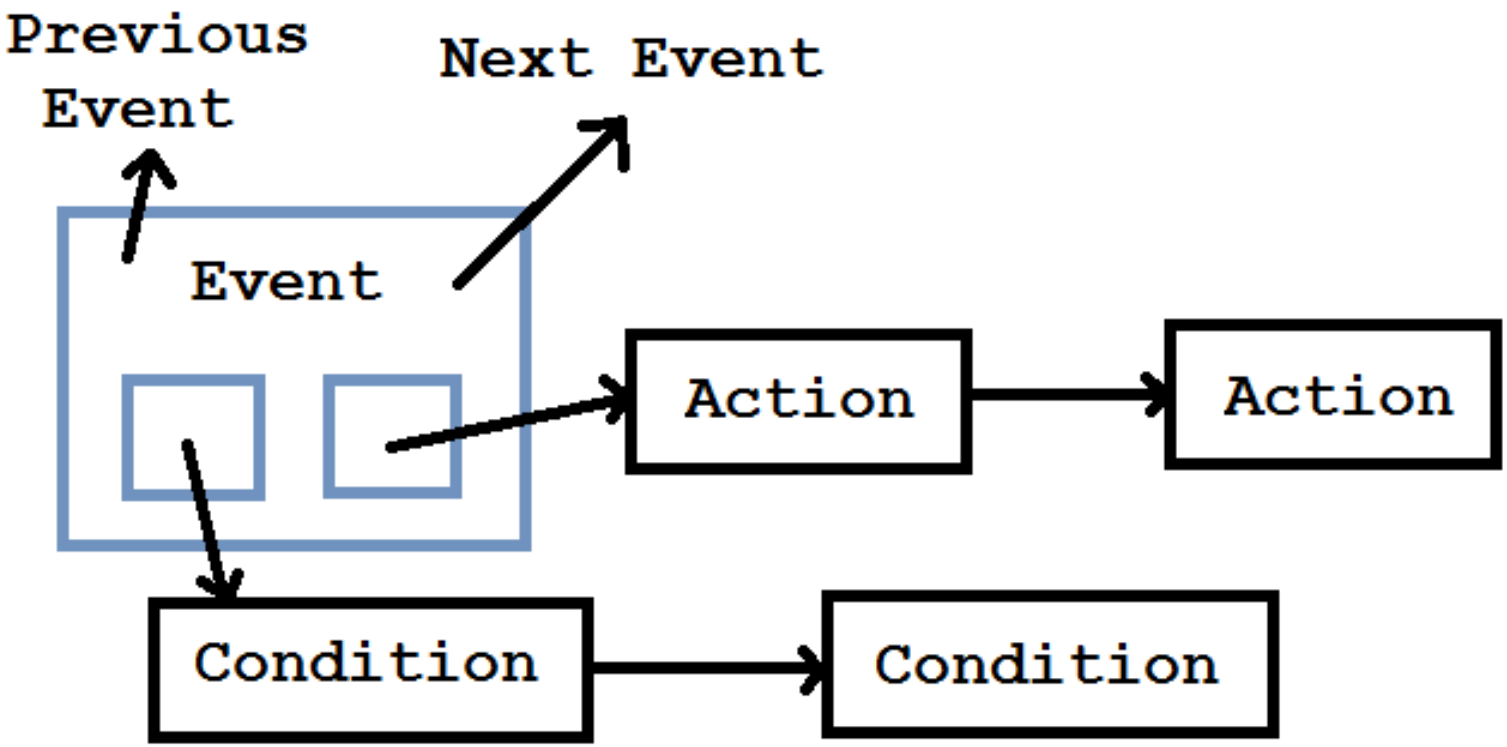
```
Conditions
  Always
Actions
  Print "Welcome to the game!"

Conditions
  TimeElapsed 5
Actions
  Print "Please select your character."
```

Implementation

- Each condition can have two parameters
  - Params come directly after name
    - Strings, numbers, names, etc.
- Each action can also have two parameters
- Each event can hold any number of conditions or actions
  - Store conditions and actions in a linked list within the event itself
- Link all events together

Implementation



Questions?

Implementation

- Updating event list:

```
for each event in the event list
  run all conditions
  if all conditions return true
    run all actions
    remove event from event list
    cleanup all conditions/actions
```

Implementation

- What about events that persist?
  - Want to run actions more than just one time
  - Usually an event is deleted once it runs actions
  - Use a “preserve action”

```
Conditions
  Always
Actions
  Print "printing a string"
  Preserve
```

Implementation

- New pseudo code for updating event list

```
for each event in the event list
  run all conditions
  if all conditions return true
    run all actions
    if no preserve action
      remove event from event list
      cleanup all conditions/actions
```

Load Events from File

- Open file and look for “Conditions” string
  - fscanf(fp, "%s%*c", buffer);
  - strcmp(buffer, ”Conditions”) == 0
- Read condition name
  - Name associates with integer index
  - Save this index to lookup functions

```
const char *ConditionArray[] = {
  "Always",
  "TimeElapsed",
  NULL,
};
```

Function Pointers

- Each condition needs to:
  - Run boolean check
  - Read in parameters when loading from file
  - Cleanup after itself (free things)
- Setup arrays of function pointers
  - One array for each of the three things above

Load Events from File

- Now that you have the index
  - Lookup function pointers (use typedefs for ptrs)

```
typedef int (*ConditionCB)( struct Condition *self );
typedef void (*ReadConditionParams)( FILE *fp, struct Condition *self );

const ConditionCB ConditionPtrs[] = {
  Always,
  TimeElapsed,
};

const cleanup ConditionCleanups[] = {
  NULL,
  NULL,
};

const ReadConditionParams ReadCParamPtrs[] = {
  ReadParamsAlways,
  ReadParamsTimeElapsed,
};
```

Load Events from File

- Order of function pointer arrays must be maintained
- How it works:
  - Get index
    - By match string
  - Save index and use in function pointer array
  - Example: assign the boolean check of condition:

```
const char *ConditionArray[] = {
  "Always",
  "TimeElapsed",
  NULL,
};
```

```
const ConditionCB ConditionPtrs[] = {
  Always,
  TimeElapsed,
};
```

Load Events from File

- Same thing with actions
  - Array of names
  - Array of ActionFunctions
  - Array of ActionCleanups
  - Array of ReadActionParams
- Assign function pointers while loading from file
- See sample code on distance for more details

Questions?

Types of Conditions

- Condition variables:
  - Counters; Factions, Locations (AABB or Circle); Timers; Object Count; Inventory (acquire enough money?); Kill counts; Scores, etc.
- Condition comparisons:
  - At least; exactly; at most
- Condition examples:
  - [Faction] bring [comparison] [number] of [type of object] to [location]
  - Player kills [comparison] [number] [type of object]

Types of Actions

- Create object(s), move object to location, modify hit points, destroy objects, setup timers, victory, defeat, etc.

Example Events

- Time elapsed is at least 10 minutes
  - Victory
- Player owns at least 5 torches
  - Turn off the lights
- Player owns at most 0 torches
  - Turn on the lights
- Player kills at least 3 villagers
  - Spawn guards around player
  - Order guards to attack player

Demo!

Final Tips

- Look at my sample code online
  - It isn't a whole lot of code
  - Can be extended easily
- Re-read the GameObject Design lecture
  - Setup global vtables
  - Reduce maintenance time to nearly zero
- Consider yourself if the whole language will save time
  - Can you code it in a reasonable time?
  - One weekend? One week?
    - If longer than a week I'd say just write C code
    - It took me two days out of a weekend, was very worthwhile!

Resources:

- Sample code on moodle
- r.gaul@digipen.edu – my email