Adventure Game Design - The Internals

A KansasFest Session by Jay Jennings

This session isn't a "hot coding" session. Writing the code is the easy part of creating an adventure game. What's more difficult is working out the internal structure of your game and figuring out how all the pieces work together. In other words, how to turn a bunch of pieces into a game that someone enjoys playing.

A History Lesson

The examples used in this session come mainly from the things learned while programming version 1.0 of the Softdisk Adventure Machine (by Eric C. Mueller and Jay Jennings) and version 2.0 (by Jay Jennings).

Other ideas sprang into existence while writing a series of articles for 8/16-Central magazine, and while programming a series of 8-bit adventure games (The Northern Nexus, Mythryal Genesis and Mythryal Dungeons) for Softdisk Publishing.

My over-riding concern while creating adventure games for Softdisk has been to create a shell that can be used over and over again by plugging different data into it. This means that code size and execution speed takes a back seat to being generic.

SAM v1.0 was script-driven. Each adventure was written with a word processor. The resulting text file was then scanned by SAM and turned into tables internally. SAM 2.0 uses an entirely different (and much better) approach. Each room, object, npc, and event is a separate resource. By doing a LoadResource call I have all the information I need at my fingertips with no parsing to worry about. A desktop-based game skrypter is available to help create SAM 2.0 adventures.

The first two versions of SAM are desktop-based. I'm starting a new adventure game series this fall that will be "normal" graphics-based. The exterior will be completely new but the SAM internals will still be used. Well, a super set of the SAM internals, anyway. I keep thinking up new ideas to implement all the time.

Jay's Game Design Philosophy

The following are some thoughts and feelings on different aspects of adventure game design. If you think differently, feel free to attempt to change my mind. I'm open to discussion on just about anything.

• Where do I go from here? If I walk into a "real" room I can immediately see the obvious exits. I don't walk into three walls before finding the door. Tell (or show) the player where they can go.

• Don't require me to do useless things. You don't make me type "Stick arm out" before picking up an object, so don't make me walk over to a sign to read it. If I want to use an object, don't require me to pick it up first.

• If you want to write a guessing game, write a guessing game, not an adventure. The "I don't know what a lamp is" crap isn't fun, it's annoying.

• The previous item relates to interface issues. I prefer to take typing out of my games altogether. As soon as you add that element you run into the problem of people typing things you know nothing about. Give people a list, either spelled out or with icons, and let them manipulate those.

• Don't allow things to happen that will make the game unwinable. It's not fair to the player. I did that in JuggleQuest -- oops!

• Create the elements of your adventure world first and then figure out your puzzles and quests based on what's possible. Do as little puzzle-specific coding as possible.

Game Particles

There are three main areas you need to be concerned about when creating an adventure game. Rooms, or the different locations in your game, objects, which includes treasures, weapons, etc., and NPCs, or non-player characters. NPCs include the bad guys (monsters) as well as townspeople, etc.

There's also one other area we need to think about -- Events. These are the things that tie the previous three items together and enable you to create a cool game. You can create a neat game without events, but not a cool game.

Rooms

• Rooms are locations in the game. A kitchen is a room as is a mountain meadow.

• Rooms are very passive. They know little more than who they are and where their exits lead. Is there a treasure chest in the room? The room not only doesn't know, it doesn't care.

• I think rooms are about 10 feet on each side. Multiple rooms can be fitted together to make one big room in the adventure.

• Rooms know if they are lit or not. If not, and if you're not carrying a lit light source, you can't see anything.

• Sounds can be played upon entering a room. A better method than just specifying an rSoundSample to play would be to point to a resource that would include not only the sound to play, but more information as well, such as whether to loop the sound. This is also true of room pictures (isn't hindsight wonderful?).

Objects

• Objects are smart little things. One of the pieces of knowledge they have is their location. They always know where they are, including in your possession. Your inventory is nothing more than a room with an ID of zero.

• Not every object is lying around in plain sight. Some things have to be searched for.

• In a perfect world (or a badly designed game) we can carry everything we come across. By giving every object a weight you can make things a little more realistic.

NPCs

• NPCs are either hostile or friendly. If they're mean they'll attack you on sight and the only thing you can do is fight them or run away. If they're friendly you can talk to them in order to glean information that might be useful.

• NPCs that talk are useful in giving the adventurer information necessary in solving the adventure. Speech can be one-time only to give the player a good reason to pay attention.

• Talking can also be optional depending on the actions of the player. The old woman and the hungry dog is a good example of conditional talking.

• Hostile NPCs do double duty as locked doors. The key needed to unlock the door is specified as the weapon needed to "kill" the NPC.

Events

• Almost every aspect of rooms, objects and npcs should be tweakable via events.

• Event commands available in SAM v2.0:

Message
Activate
Deactivate
Chain to Event
North Door
South Door
East Door
West Door
Up Door
Down Door
Show Picture
Show Modal Picture
Play Sound
Change Health
Change Current Room
Change Use Event
Change Use Condition

• Example: Using events to switch items behind the player's back (such as when duplicate treasure chests are needed).

• Example: A king sends you on a quest and only allows you to marry his daughter when you've accomplished the quest.

• Example: Chaining events for more power or to give hints. Old woman and dog, two talk events chained (second one created inactive).

• Adding a "Chain Both Parms" command would allow things to be even more flexible, as would a specific "chain to" field in the event structure.

Other Cool Things We Could Do

• Speech Trees for NPCs

Want to talk to an NPC? They only have one thing to say right now. By implementing a speech tree every NPC can know a different subset of the knowledge in the land. The town drunk knows very little, the ancient wise man knows quite a lot. Speech may be a reason to disregard my "no typing" rule. Events can also be used to switch duplicate NPCs. This would allow the "same" NPC to say different things at different times.

• Varying Levels of Hostility
Currently there are hostile and friendly NPCs in my games. That will change in my next series. Each NPC will have a hostility level that varies from 1 (very friendly - don't bend over in the shower) to 10 (extremely hostile - kill on sight). Player actions will have some effect on NPC hostility levels.

• A Wandering We Go
An NPC is a stationary thing at this point. Being able to walk around from location to location is coming soon. This will include varying levels of walkabouts in order to keep dungeon dwellers in dungeons, clerks in town, etc.

• Player Stats
The only thing kept track of right now is the health, a number between 1 and 100. Amount of weight able to carry could depend on health, as could ability to fight a monster, find and use objects, etc. Other stats could affect how our player interacts with hostile and friendly NPCs.

• The Passage of Time
Lights burn out in the SAM games but little else happens with the passage of time. Walking from room to room could exact a time hit, as could searching a room, using an object, talking to people, etc. Friendly NPCs could disappear at night and nocturnal creatures could appear.

• Combat that makes sense.

• Hostile NPCs don't actually have to be hostile. Bippy the Clown can bar you from the circus by creating him as a locked door.

Starting the Creation Process

• Figure out your basic story line. Who's in the adventure? What's the goal? What kind of area does it take place in? Make notes of this information. It will help keep you on the right track.

• Create a map consisting of all the areas in your adventure. Besides charting the actual rooms in the adventure, be sure and mark all the doors so you know where you can go from each room. Number all the rooms and give them names. This will help you figure out where to put objects, monsters, etc.

• Create a list of treasures (give them both names and numbers) and other objects that can be found in your game. Be sure and include the more mundane items like chairs, pictures on the wall, salt shakers, etc. Specify what room each object is in.

• List all of the NPCs (give them names and numbers). Include the things like blood-

sucking viper weasels as well as the girl next door. Specify what room each NPC is in.

• Now the hard part. To create a fun adventure there needs to be puzzles for the player to figure out. These, and other actions, are events. Create a list of events (again, give them both names and numbers).

• Figure out a way to tie

(Note! At this point I was interrupted by my boss. He needed to swap my Mac II for an SE/30. By the time I got everything backed-up and restored, I'd forgotten what my last point was going to be.)

Make Room, Make Room!

Locations in an Adventure Game

```
struct roomStruct
        {
        word  version;
        long  id;
        char  name[26];
        long  north;
        long  south;
        long  east;
        long  west;
        long  up;
        long  down;
        word  lit;
        long  entryEvent;
        long  searchEvent;
        long  text;
        long  picture;
        long  sound;
  };
```

Version: Earliest version of SAM needed to use this room. This is used so later, more advanced game shells will know how to handle this room.

ID: An ID number between 10,000 and 19,998 that is unique to this room.

Name: The name of this room (shown in a window title bar in the game).

Directions: These show the rooms found if you move in a specific direction from your current location. The room ID numbers are used in these fields.

Lit: Is the room normally lit or dark? This can't be changed during game play. In other words, it's either always day or night in the adventure.

Entry Event: Should an event be triggered when the adventurer comes into this room? If so, the ID of the event is found here.

Search Event: If an event should be triggered when this room is searched, that event ID is found here.

Text: The ID of the description of the room (rText) is found here.

Picture: ID of a picture to show when the adventurer is in this room.

Sound: A sound to play when the adventurer enters this room.

Sugar and Spice and Everything Nice

Objects in the Adventure

```
struct objectStruct
        {
        word  version;
        long  id;
        char  name[26];
        word  active;
        word  weight;
        word  hidden;
        word  weapon;
        long  room;
        word  points;
        word  light;
        word  consume;
        word  count;
        long  lookEvt;
        long  lookCond;
        long  getEvt;
        long  getCond;
        long  dropEvt;
        long  dropCond;
        long  hitEvt;
        long  hitCond;
        long  useEvt;
        long  useCond;
        long  openEvt;
        long  openCond;
        long  closeEvt;
        long  closeCond;
        long  text;
        long  picture;
        long  sound;
  };
```

Version: Earliest version of SAM needed to use this object. This is used so later, more advanced game shells will know how to handle this object.

ID: An ID number between 20,000 and 29,998 that is unique to this object.

Name: The name of this object (shown in a list in the game).

Active: If an object is active you can do things with it. By messing with this flag you can make objects appear and disappear in the game.

Weight: By giving an object a specific weight you keep the player from picking up everything in the game. Objects with a weight of zero are not movable.

Hidden: If the object is hidden it will remain out of sight until the room it's in is searched.

Weapon: A non-zero value says the object can be used as a weapon. The higher the number, the more powerful the weapon is.

Room: What room is this object in?

Points: The number of points a player gets for picking up this object. Not used in SAM at this time.

Light: A zero value means this object is not a light, a one says it's a light that's currently shedding light, and a two says it's a light source that's off.

Consume: Zero means this object cannot be consumed. A one means it can be consumed and disappears, a two means it can be consumed but stays around afterwards.

Count: This is the number of turns a light stays on, or the number of times an item can be consumed, depending on whether it's a light or consumable object, of course.

Actions: When we do one of the actions (look, get, drop, hit, use, open, close) we can trigger an event. We can also set a condition that must be met before the event is triggered.

Text: The ID of the description of the object (rText) is found here.

Picture: ID of a picture to show when this object is looked at.

Sound: A sound to play when the adventurer looks at this object.

Lions and Tigers and Clerks, Oh My!

Non-player Characters in the Adventure

```
struct npcStruct
        {
        word  version;
        long  id;
        char  name[26];
        word  active;
        word  hostile;
        word  strength;
        word  speaker;
        long  room;
        long  killedWith;
        long  lookEvt;
        long  talkEvt;
        long  hitEvt;
        long  diesEvt;
        long  killsYouEvt;
        long  text;
        long  picture;
        long  sound;
  };
```

Version: Earliest version of SAM needed to use this NPC. This is used so later, more advanced game shells will know how to handle this NPC.

ID: An ID number between 30,000 and 39,998 that is unique to this NPC.

Name: The name of this NPC (shown in a list in the game).

Active: If an NPC is active it can do things. By messing with this flag you can make NPCs appear and disappear in the game.

Hostile: This flag tells us whether the NPC is mean or friendly. A hostile NPC will attack us at first sight, keeping us out of any area it's in.

Strength: A three digit number that specifies how strong this NPC is. This is the value that's decreased when you battle an NPC.

Speaker: If the NPC can speak this flag is set. Obsolete at this time so the space is available for something else.

Room: The current location of the beast.

Killed With: The ID number of an object that must be used in order to kill the NPC. In the case of a locked door, this would be the key.

Action Events: When you do a specific action on an NPC an event can be triggered. Also, if the NPC kills you or you kill the NPC.

Text: Description of the NPC.

Picture: Picture of the NPC. If the NPC is hostile this shows up when you're attacking it. If the NPC is friendly, it shows up when you look at or talk to the NPC.

Sound: Sound that's played when you come in contact with the NPC (or when you look at a friendly NPC).

What's Happening, DØØd?

Events in Our Adventure

```
struct eventStruct
        {
        word  version;
        long  id;
        char  name[26];
        word  active;
        word  flag;
        word  what1;        /* the first command */
        long  spec1A;
        long  spec1B;
        word  what2;        /* the second command */
        long  spec2A;
        long  spec2B;
        long  text;
        };
```

Version: Earliest version of SAM needed to use this event. This is used so later, more advanced game shells will know how to handle this event.

ID: An ID number between 40,000 and 49,998 that is unique to this event.

Name: The name of this event (used only in development).

Active: If an event is active it can be triggered. By messing with this flag you can make events appear and disappear in the game.

Flag: The only function of this flag at this time is to specify whether the event is a one-time happening.

First Event: The first of two commands that happen when this event is triggered.

Parameters: Up to two parameters used by the above command.

Second Event: The second of two commands that happen when this event is triggered.

Parameters: Up to two parameters used by the above command.

Text: The ID of text that should be shown when this event is triggered..