Tree Language Virtual Machine Design Document

Scopes

Overview

Global

Code and data shared by all objects in the game, like utilities and environmental parameters, and the objects that model the world. This includes common utility functions and the simulator global state, language and user interface data, as well as the world grids, house, family, and object module.

Semi Global

Code and data shared by related objects like chairs, doors, and people. This includes utilities used to implement common protocols like sitting down and walking through.

Private

Code and data stored in the objects file, shared by all instances of other objects in the same file. This includes actions and tree tables particular to one object, or the parts of a multi tile object in the same file.

Object

Data stored with each instance of an object that's instantiated in the world, which is saved in the scenario file. This includes normal objects, as well as people (who have even more state). Every object has a stack, temporary variables, attributes, simulation data, object definition, globally unique id, semi global file, private file, relationship matrix, location, and other properties.

Local Stack

Each object has a stack containing state that is pushed and popped between calls to trees. Each invocation of a tree has its own stack frame, which contains the context, program counter, stack object target id, interrupt flag, and stack variables.

Details

Global

There is a global resource file that contains data shared by all objects, called the "Language File".

There is another global resource file that contains the application's users interface resources and other data, called the "GUI File".

The simulator maintains global state accessable to tree programs in a segment called the "Sim Globals", as well as other state like the funds, running state, and tick count.

The Sim Globals include the hour, day of month, temperature, weather, funds, minute, second, month, year, current family, and current house. Trees can read and write them directly.

The simulator also maintains a great deal of state in the World grids, House object, Family object, and Object Module.

The World grids model the terrain altitude, ground cover, floor pattern, room numbers, walls, doors, windows, and object placement lists.

The House object maintains the family, object module, world grids, simulator, save file name, and house number.

The Family object contains a list of people, the family name, family number, and house number.

The Object Module keeps track of the error file, language file, 3d device, meshes, and all objects in the world.

Semi Global

Each object can have an optional Semi Global file, that is shared with other similar objects in different files. These include common utilities that implement shared protocols between objects like generic people behaviors, sitting down in chairs, walking through doors, etc.

Private

All instances of a particular object share private data from the object's file. One file can contain several related objects (represented by "Object Selectors"), like the different parts of a multi tile object.

An "Object Selector" refers to code and data that belongs to the class of objects, not individual instance data. A file can contain several related object selectors, but usually only has one.
An object selector refers to the private file from which it came, a semi global file that is shared with other object selectors, a renderer, a behavior, a file name, an object name, an animation table, a header, a globally unique id, a definition resource id, and a unique index that distinguishes it from other object selectors loaded into the current runtime.

An object file can contain object definitions, trees, tree tables, icons, draw groups, animations, suites, and sounds.

An object definition keeps track of the graphics, behaviors, tree tables, personality, type, animation table, globally unique id, price, slots, flags, and other properties of a class of object. An object definition includes the following state of 22 numbers.

- version
- stackSize
- baseGraphic
- numGraphics
- initBhav
- toolbarPict
- treeTableID
- personalityID
- type
  - kUnknown, kFood, kPerson, kContainer, kFurniture, kStructure, kAnimal, kSimType,

kDoor, kMouseType, kUserAvatar, kInternal
masterID
subIndex
dialogID
animTableID
guid
disabled
portalTreeID
price
bodyStringsID
slotsID
headLinesID
eventTreeID
selfModTreeTableID

Object

Each instance of an object has its own private object state, that is saved in the scenario file.
Types of object include inanimate objects as well as people, who have more state than inanimate objects.

Each object has a reference to an object definition, that is shared by all objects of the same class, and comes from the object's private file.
It defines the class type and behavior of all instances of that class.

Every object has a stack for local storage of nested tree parameters, as well as temporary variables, numeric attributes, magic simulation data, a globally unique id, an object definition, a relationship matrix, a list of slots, as well as other miscellaneous and indirect properties.

The stack is pushed and popped between calls to trees, and is described below.

The 8 temporary variables are shared between calls to trees, and can be used to pass parameters.

The 8 numeric attributes can be used by tree tables to model the object's state, and are not touched by the simulator.

The magic simulation data is maintained and referenced by the simulator and the user interface, and can be read and changed by the tree tables. The simulation data includes the following state of 36 numbers.

kGraphicNumber
kDirection
kColor1
kColor2
kPattern
kHeight
kRouteID
kIndirectID
kFlagField1
kCanContain
kCanWalkOver
kCanWalk
kPreviouslyFound
kOccupied
kNotified
kRoutingInterruptable
kAnimID
kAnimFrame

```
kObjectID
kOldTargetID
kWallPlacementFlags
kSlotID
kFamilyNumber
kUnused5
kCounter1 = kTrapCount = kRoomCompDelay
kAge
kGender
kTreeTableEntry
kSearchRadius
kSpeed
kRotationSpeed
kCounter2 kRouteCount
kUseCount
kContainerID
kWeight
kSupportWeight
kRoom
kRoomPlacement
        kRmPlAnyRoom
        kRmPlOutsideOnly
        kRmPlInsideOnly
kUnused2
kUnused3
kUnused4
kHidden
```

The relationship matrix is two dimensionally indexed by the guids of other object and relationship identifiers, and it stores numeric values used by the trees to keep track of how individual objects relate to other individual objects and classes of objects.

Each object can have a list of slots enumerated in the object definition. An object's slots define relative locations, so each object may have other objects contained in its own slots.
Slots are used for containing other objects, sitting on or entering other objects, carrying an object by a registration point, and placing headlines.

Local Stack

Each object has a stack of a fixed depth, that is pushed and popped between calls to trees, to keep track of nested tree invocations and local storage. Each stack frame contains the context a program counter context (consisting of a behavior pointer, tree id, and node number), a target object id (an implicit object reference like "this" in C++, that the tree program can change), an interrupt flag (used to restart blocked primitives), and four short local stack variables.

When a tree is called as a subroutine, the local stack variables are initialized from either the constant parameters of the call, or the temporary variables. If the constant parameters are the default value of -1, -1, -1, -1, then the first four temp variables are copied into the stack frame's locals instead.

Person

A person is a subclass of an object, that contains additional state and behavior. Each person has an XVitaBoy reference, a Personality, an XAnimator, a set of Motives, an Action Queue, a Headline, a Skeleton Name, a Body Suit Name, a Head Suit Name,

some standard Slots, and a set of Actions.

The motives consist of the following numeric properties, ranging from -50 to 50.

HappyLife, HappyWeek, HappyDay, HappyNow
Physical, Energy, Comfort, Hunger, Hygiene, Bladder
Mental, Alertness, Stress, Environment, Social, Entertained

The Personality is a set of numbers that control how an individual's motives interact with the environment, but it's not used yet.

The Action Queue is a list of pending actions, so the user can click ahead to get a person to do several things in a row. There is an issue that you may not be able to select an action until its preconditions have been satisfied, so you may have to wait before being able to click ahead. Another problem is that you can click ahead an action that might be disabled by the time it's executed. This needs to be worked out.

The XAnimator and XVitaBoy objects wrap the lower level graphics and VitaBoy objects, and implement higher level behavior like animation channels and walking behaviors.

The Skeleton, Body, and Head names identify the VitaBoy resources to use for the animated character. The person can have one of the following heads: Default, Happy, Sad, Mad, Sleep.

People have the following standard slots: Right hand, Headline.

Each person has a headline slot where thought balloons and emotional icons are displayed. These can come from the global file or the person's private or semi global file.

Motive though balloon icons include the following:

None, Energy, Hunger, Comfort, Hygiene, Bladder,
Stress, Alertness, Entertained, Social, Environment

Headline icons include the following:

None, Stress, Smell, Hurt, Drunk,
Love, Idea, Suprise, Hate

People have the following standard feedback animations:

Stand, Hunger, Sleepy, Wired,
Bladder, Mad, Entertained

Each person also have a standard set of personal stylized walking and standing animations.

Trees

A tree is a program that controls the behavior of an object or person. Trees are actually hierarchically nested directed graphs of nodes. Each tree contains a graph of nodes, strung together into a spaghetti state machine.

A node can pop the stack and return true or false from the tree, or it can contain a reference to a primitive or a tree, as well as four numeric parameters, and true and false transitions that point to other nodes in the same tree.

When a node pops the stack returning true or false, the interpreter returns to the calling tree and follows the corresponding true or false transition.

When a node refers to a primitive, it is called with the parameters as arguments, and it can refer to and modify the state of the virtual machine and the world.

When a node refers to another tree, it is called recursively by pushing another stack frame, and passing arguments (from either the parameters or the temporary variables).

The result of a call to a tree or a primitive may return true or false, which determines the next node in the tree to transition to and execute next. A node may have just one transition, or both a true and a false transition. Transitions refer to nodes in the same tree.

There are several types of trees, including Generic, Portal, Check, Container, and Switch. Generic trees are the most common, and when they are called, execution begins with the first node in the tree (which is displayed with a green border around it). Switch trees take an argument, and start execution with the node indexed by that parameter, so if the parameters is 0, the first node is executed, and so on. Other tree types are not so common or useful to tree programmers.

Each node of a tree has an optional comment, where the programmer should describe its meaning in English. Trees also have a list of comment windows for overall documentation and rationalization.

The execution of a tree or a primitive can cause an error, which is signaled with a debugger window. There are problems with debugging errors that happen in many circumstances, like check trees, that need to be worked out.

A node can have a breakpoint set for debugging purposes. This isn't actually stored in the node, and has some problems and restrictions that need to be worked out.

Tree Tables

Tree tables are used to define a set of actions with precondition tests, that can advertise that they satisfy various motives (although they don't actually have to satisfy the motives they advertise). The menus that pop up on objects are defined by tree tables. A tree table consists of a list of tree table entries. Each entry has an optional check tree id, which is executed as a precondition to tell if the entry is selectable. If the check tree returns true, the corresponding menu item is enabled, otherwise it's disabled. Each entry has an action tree id that is executed if the precondition returns true and you select that menu item. Each entry also has a set of motive advertisements, that default to 0. If an advertisement is non-zero, it means that the tree table entry might satisfy the corresponding motive by the given amount. Tree table entry motive advertisements combined with check tree preconditions are used together to implement chaining motives, so a person who's hungry goes to the fridge to get food, cooks the food, puts it on the table, and eats it, because the advertisements lead him through the task like a carrot.

Primitives

Here is a list of the currently implemented primitives.

kIdle
kSearch

```
                kAttr
                kGoto
                kGrab
                kDrop
                kStartSearch
                kUpdate
                kRandom
                kDeltaMove
                kAnimate
                kDistanceTo
                kDirectionTo
                kPushAction
                kAlert
                kTreeBreak
                kNeedExpr
                kIdleForInput
                kKillObj
                kFindTreeNew
                kUnusedPrim2
                kUnusedPrim3
                kUnusedPrim4
                kPlaySound
                kRelVar
                kSpendMoney
                kAnimateOneLoop
                kGotoRel
                kSpeak
                kTreeSearch
                kGosubTree
                kGetNextObject
                kConstructMaxTree
                kFind5WorstMotives
                kIncrementNeed
                kGetMotive
                kShowDialog
                kGetFrontObject
                kRunTypeTree
                kShowHide
                kSetHeadline
                kSetThoughtBalloon
                kCreateObject
                kDropOnto
                kAnimateNew

    Primitive Parameters
            Data Owners
                    Many primitives allow you to select different argument
                    address modes for their parameters, called data owners.
                    These include the following.
                            kMyself,                // my own object's attribute block
                            kTreeParam,             // the object of this behavior
    tree's attr block
                            kTargetObj,             // the target object's attr block
                            kMyData,                // my own object's data
                            kTreeParamData, // the tree's object's data
                            kTargetObjData,
                            kSimGlobals,    // simulation globals
                            kImmediate,             // just plain old data
                            kTempVars,              // temporaries of my object
    (referenced by number)
                            kStackVars,             // stack parameters
                            kStackObject,   // the object on the stack at any level
                            kTempTempVars,  // temporaries of my object (referenced by
    another temporary)
                            kStackObjTreeTableAdvertisement, // the array of floats in
    stackObject->GetTreeTable()->GetEntry(fData[kTreeTableEntry]);
```

```
                        kTreeParamTemps,
                        kPersonMotive,
                        kStackObjectMotive,                     // the literal
motive of stack object
                        kStackObjectSlot,
                        kStackObjectMotiveOfTemp,       // the motive of stack
object indexed by a temp



ResFiles
        There are two global resource files:
                Gui file
                Language file

        There are many different classes of resource file:
                iResFile
                FlatResFile
                IFFResFile
                MacResFile
                MultiResFile
                ResolveResFile
                NResFile
                SeqResFile
                ObjResFile
                ChainResFile
                        Private
                        Semi global
                        Global
```