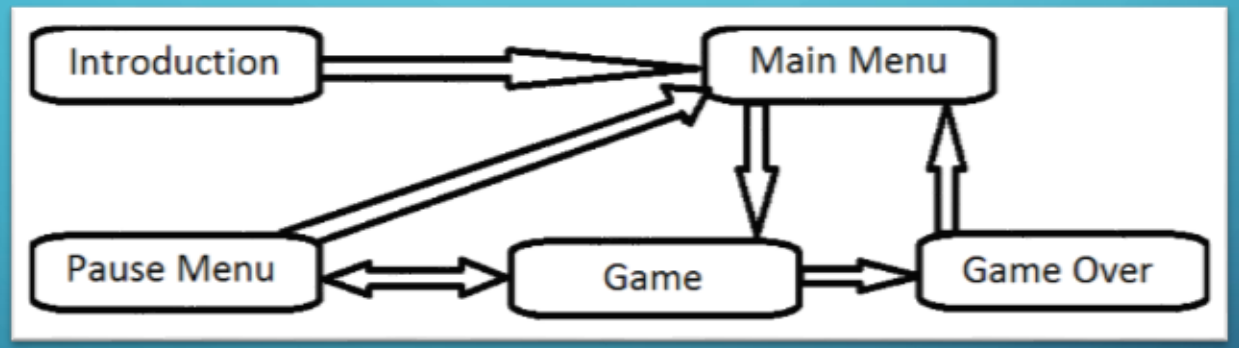
AN ARCHITECTURE FOR GAME STATE MANAGEMENT BASED ON STATE HIERARCHIES

JUAN DANIEL LASERNA CONDADO

STAGES OF THE GAME

- Games have different stages



Example of a game state machine

IMPLEMENTATION OF GAME STATE MANAGEMENT

- Uses a Game Handler
- Assign identification numbers (ids) to the states
- Is common specially with non-object-oriented languages

```
#define INTRO      0
#define MAIN_MENU  1
#define PAUSE_MENU 2
#define GAME       3
#define GAME_OVER  4

void GameHandler()
{
  switch (GetCurrentState())
  {
    case INTRO: Intro(); break;
    case MAIN_MENU: MainMenu(); break;
    case PAUSE_MENU: PauseMenu(); break;
    case GAME: Game(); break;
    case GAME_OVER: GameOver(); break;
  }
}
```

PROBLEMS

- The main function handler can become very long
- It is difficult to maintain as the number of states increases
- The game state handlers potentially replicate the game loop implementation making it difficult to change it later
- Bad scalability (increase their complexity and maintenance cost)

IMPLEMENTATION OF GAME MANAGER

- Uses a Game Manager class to keep track of the current game state class
- Every state derive from an abstract class and they are designed as a singletons
- The Game Manager presents a method to change the current state which accepts object instances
- This solution implements temporary and definitive state transitions
- ***It do not take into account state hierarchies***

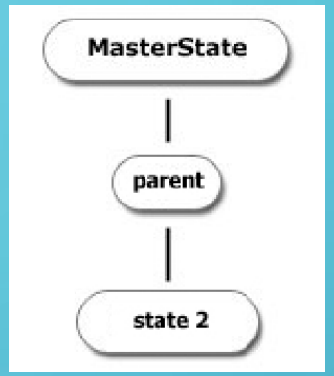
SIMPLE COOPERATIVE MULTI-TASK SYSTEM DESIGN AROUND A PROCESS CONCEPT

- A process is described as an execution unit and is designed as a pure virtual class with a single method named *Update()*
- This concept is excessively general and is at a lower-level than game states

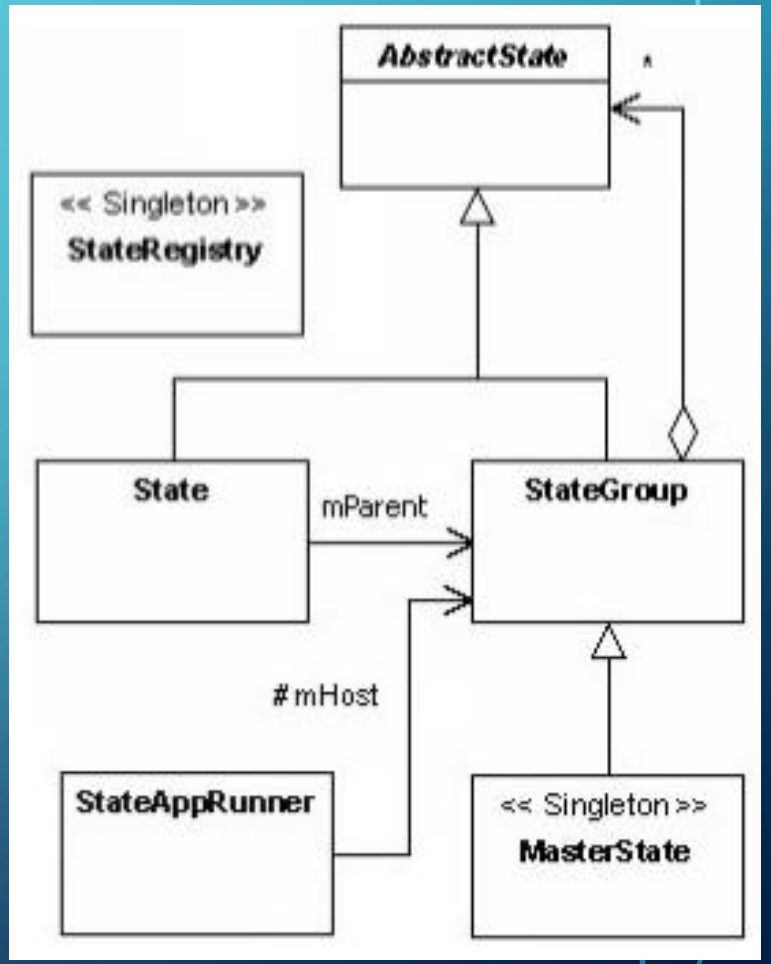
THE GUFF FRAMEWORK

- This tool is divided into two main modules:
  - The application layer is modeled as a state machine
  - The toolkit comprises sets of ancillary classes which handle visualization, automatic management of third party libraries, application configuration, input devices, Math, audio, and utilities
- This tool is an open source project
- Last update was on August 1, 2007

THE GUFF FRAMEWORK



- All the states must have a parent
- The default state is called MasterState
- Any referenced state must have been created previously in order to be used
- The only state that does not have a parent is the masterState singleton



THE GUFF FRAMEWORK

- **The AbstractState class**
  - It is the common interface for all states in the application layer
  - It has three types of operations/events
    - **System events**
    - **State events**
    - **Events related to the game loop execution model**

THE GUFF FRAMEWORK

- **The AbstractState class > System events**
  - **OnWindowResize:** Window's size changes
  - **OnMouseDown, OnMouseUp, OnMouseMove:** Mouse events
  - **OnKeyDown, OnKeyUp:** Keyboard events
  - **OnActivate, OnDeactivate:** The application gains or loses focus

THE GUFF FRAMEWORK

- **The AbstractState class > State events**
  - **OnInit:** Dispatched at application startup, on behalf of all states. This is an opportunity for all states to perform whatever initialization is required for their operation
  - **OnShutDown:** Dispatched at application shutdown, so the states are able to release their resources back to the system
  - **OnEnter:** Dispatched whenever the application enters the state
  - **OnExit:** Dispatched whenever the application leaves the state

THE GUFF FRAMEWORK

- **The AbstractState class > Events related to the game loop execution model**
  - **OnFixedFrequencyUpdate:** Represents the fixed frequency update stage
  - **OnUpdate:** Represents the update stage
  - **OnRender:** Dispatched whenever it is necessary to render an image

THE GUFF FRAMEWORK

- **The StateRegistry class**
  - This class applies the singleton pattern
  - This class serves as a global state This class serves as a global state This class serves as a global state repository
  - Its main responsability is to map names to state instances
  - It is required that the states have unique names

THE GUFF FRAMEWORK

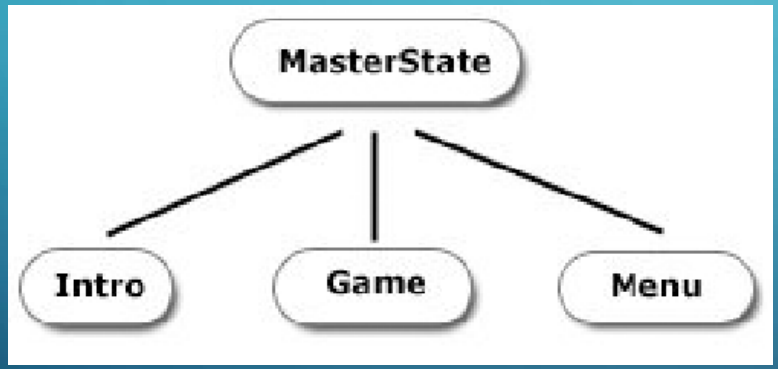
- **Definitive changes**
  - void changeState (const string &stateId);
- **Temporary changes**
  - void pushState (const string &stateId);
  - void popState();

THE GUFF FRAMEWORK

- When a temporary state transition is requested, the state group pushes the current state into the stack and performs the operation. Next, when the state finishes its purpose, the group is able to restore the former state by popping it from the stack. Definitive state transitions do not save state information into the stack.

THE GUFF FRAMEWORK

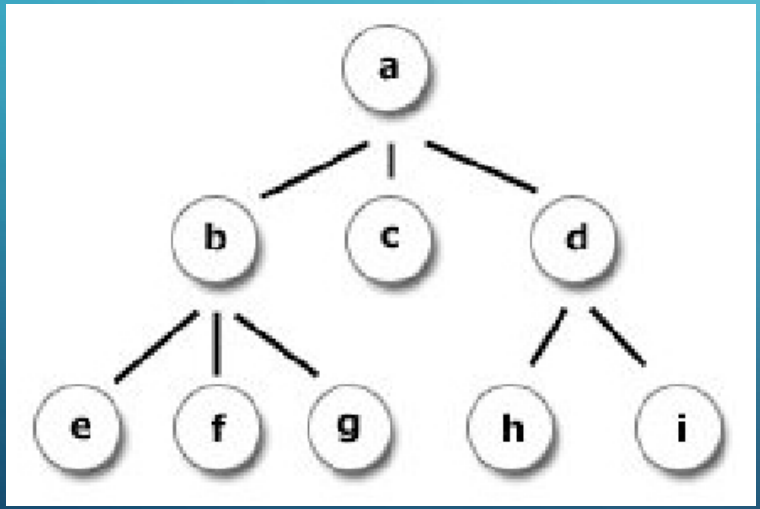
- **State transition implementation details**




```
void Intro::OnUpdate (float time)
{
 totalTime += time;

 if (totalTime > 10)
 {mParent->changeState ("Game"); return;}

}

void Game::OnUpdate (float time)
{
 command = parseInputData ();

 if (command == MENU)
 { mParent->pushState ("Menu"); return; }
 ...

}

void Menu::OnUpdate (float time)
{
 command = parseInputData ();

 if (command == EXIT_MENU)
 { mParent->popState (); return; }
...

}
```

THE GUFF FRAMEWORK

- **State transition implementation details**




```
void StateGroup::changeState (const
string & stateId)
{
 if (theStateIsMine (stateId) )
  doChangeState (stateId);
 else
 {
   if (mParent != 0)
    mParent->changeState (stateId);
   else
   // invalid transition, an error is
   // reported
 }

}
```

THE GUFF FRAMEWORK

- **State transition implementation details**

```
void StateGroup::doChangeState (const
string & stateId)
{
 if (stack is not empty)
   (top of the stack)->OnExit ();

 emptyStack();
 put stateId on top of the stack;
  (top of the stack)->OnEnter ();

}
```

GAME IDEA

- Title: Maxigame
- Genre: Minigames
- Reference game: Crash Bash

MAXIGAME

REFERENCES

- Valente L., Conci A., Feijó B. (2006), An Architecture For Game State Management Based On State Hierarchies
- LAMOTHE A. (2003), Tricks of the 3D Game Programming Gurus. Indianapolis: Sams Publishing
- VALENTE L. (2005) Guff: Um sistema para desenvolvimento de jogos. Master's thesis, Universidade Federal Fluminense [in Portuguese]
- GUFF PROJECT (2006) Guff project home. Available from: http://guff.tigris.org [Accessed 20 August 2006]
- GAMMA E., HELM R., JOHNSON R., VLISSIDES J. (1995) Elements of Reusable Object Oriented Software. Reading: Addison-Wesley
- VALENTE L., CONCI A. AND FEIJÓ B. (2005) Real-time game loop models for single-player computer games. In: Proceedings of the IV Workshop Brasileiro de Jogos e Entretenimento Digital, 23-25 November 2005 São Paulo. Porto Alegre: SBC, 89-99
- LEWIS T. (2003) Managing game states in C++. Available from: http://gamedevgeek.com/tutorials/managing-gamestates-in-c/ [Accessed 20 August 2006]
- LAROCQUE D. (2001) State pattern in C++ Applications. Available from: http://www.codeproject.com/cpp/ statepattern3.asp [Accessed 20 August 2006]
- MCSHAFFRY M. (2003) Game Coding Complete. Scottsdale: Paraglyph Press