Failures in game development and technology

Failures in game development have rarely been linked to issues of the development framework; the motivation link between Casanova and productivity is weakened in this respect

- The game **Star Wars: The Old Republic** encountered steep difficulties because of its engine. From Bioware's James Ohlen talk at GDC 2013: *Prior to launch, Star Wars had problems. [...] The HeroEngine in particular had extra problems; [...] All the basic elements of an MMO were a struggle. Building the auction house took four months and the best programmers in the company; the basic features of chat, guilds, PvP and others were still being developed throughout 2011 and took time away from innovation – meaning the team reserved their efforts for the story. This too caused problems, with the branching paths raising development costs even further, much more than the seemingly-expensive voice recording.*
- From a Gamasutra publishers poll: the highest cost in making games costs from developers. In particular, programmers are the most expensive employees overall. Technology is thus the central expense in making games.

Prototyping

The link between prototyping and Casanova is not explored.

- Casanova is highly suitable for prototyping, thanks to
  - rapid, highly effective development
  - less attention to technological details
  - readable, high-level resulting code; editable by designers (apparently, given some informal experiments)
  - focus on automated optimization rather than manual optimization
  - possible integration with more sophisticated game/physics engines

To the bare metal

How much does it take to go from the high-level to the bare metal?

- Compilation process generates (inlined) functions for:
  - update and draw
  - game loop
  - coroutines
  - double buffering is implemented through arrays of length 2 for rules
  - double buffering of Rule¡List¡T¿¿ is done with arrays of length 2 containing preallocated chunks of memory
  - drawing uses a default implementation that batches sprites and text; other implementations may be added easily

Optimizing further

What code optimization avenues are left open to experienced developers?

- Any, even though they are not aligned with the (implicit) style of Casanova programs, which emphasizes simplicity of code and relies on automated optimizations:
  - optimizations that can fit into purely functional data structures can be applied to rules
  - other, imperative-style, optimizations can make use of `Var<T>` and scripts

Casanova and patterns

How does the rule-script-draw design pattern for games compare to existing patterns used in games?

- Patterns in game design identifies *design* patterns, unconcerned with implementation details. As such these patterns do not relate to the technical aspects of game development that Casanova focuses on. Examples of such patterns are:
  - perceived chance of success (until the end)
  - paper-rock-scissors
  - (avoiding) analysis paralysis
  - mutual/shared goals and reward

Casanova and patterns

How does the rule-script-draw design pattern for games compare to existing patterns used in games?

- Other studies, in particular *Evaluation of oo design patterns in game dev*, focus on the application of traditional design patterns to games. They find that many design patterns lend themselves well to games, for example:
  - strategy
  - state (LOD)
  - bridge (drawing a 3d model with a 2D impostor)
- Industry consensus seems (informally) that a very useful design pattern is the "component" pattern

Casanova and patterns

How does the rule-script-draw design pattern for games compare to existing patterns used in games?

- In general though, these patterns focus on some limited aspect of game development, and not on all-encompassing solutions such as those offered by Casanova. So while design patterns are a toolbox useful to build *common parts* of games, Casanova is a single tool that allows to build *whole* games.

Comparison with UDK

How does Casanova compare to UDK and its scripting languages?

- Casanova is not a scripting language, that is it allows to build a full game from the ground up. UDK is a game development engine, where a limited set of the game dynamics can be parameterized through scripts.
- Unreal script supports interaction with the UnrealEngine graphics objects, plus (excerpt from the documentation):
- *To provide Java-style programming simplicity, object-orientation, and compile-time error checking:*
  - *a pointerless environment with automatic garbage collection;*
  - *a simple single-inheritance class graph;*
  - *strong compile-time type checking;*
  - *a safe client-side execution "sandbox"*
  - *and the familiar look and feel of C/C++/Java code.*

Comparison with UDK

How does Casanova compare to UDK and its scripting languages?

- *major concepts of time, state, properties, and networking which traditional programming languages don't address.*
- *This greatly simplifies UnrealScript code.*
- *The major complication in C/C++ based AI and game logic programming lies in dealing with events that take a certain amount of game time to complete, and with events which are dependent on aspects of the object's state. In C/C++, this results in spaghetti-code that is hard to write, comprehend, maintain, and debug.*
- *UnrealScript includes native support for time, state, and network replication which greatly simplify game programming.*

Comparison with UDK

How does Casanova compare to UDK and its scripting languages?

- *To enable rich, high level programming in terms of game objects and interactions rather than bits and pixels.*
- *Where design tradeoffs had to be made in UnrealScript, I sacrificed execution speed for development simplicity and power.*
- *After all, the low-level, performance-critical code in Unreal is written in C/C++ where the performance gain outweighs the added complexity.*
- *UnrealScript operates at a level above that, at the object and interaction level, rather than the bits and pixels level*
- For added simplicity, Kismet is a visual, data-flow programming language used to build state machines that interact and generate new events. It is a limited, but very simple, layer built on top of UnrealScript.

Further assessment by comparison

Expressiveness is not assessed by comparing Casanova to other frameworks over a set of game genres

- As much as it would be desirable, such a comparison would entail:
  - building multiple games in Casanova *and* other languages and frameworks; this takes time that could be spent best on improving Casanova itself
  - using existing open source implementations is problematic, as they are often built by amateur developers and with very little guarantees in terms of quality
  - included is a comparison of tetris between Casanova, C#, Java, C++, and Python