**Reviewer 1**

Section 1:

A precise definition of the notion of encapsulation is lacking, in particular with respect to the principles of object-orientedness. This is key to understanding the issues discussed in the example of Section 3.3.

**Added definition with reference in the suggested point**

In the paragraph "What seems ideal", it would be worth mentioning the issue of debugging difficulties that may crop up when a piece of code is heavily transformed at the source or intermediate levels. The user's original code may be so much transformed that getting it to properly work may be difficult to achieve.

**Explained what doing this at source code level would imply for the developer**

At the end of Section 1 (and the paper as a whole), it would be good to try to better integrate the two parts of the paper by emphasizing that efficiency and networking issues are just two use cases testing the hypothesis that encapsulation can be preserved in Casanova 2 via optimization transformations. Maybe a title along the lines of "Encapsulation preservation in Casanova 2: performance and networking" would be more appropriate.

**Added a small explanation to better connect the networking part to the encapsulation part**

Mention that some future work is discussed in the conclusion (and beef-up that part there)

**Referenced the conclusion in the summary in the introduction and expanded the future work description in the end by mentioning further experiments we are conducting by using meta-compilation techniques to address the same problems.**

the last paragraph mentions reflection, but why is this a useful feature for game programming.?

**In this section we are explaining why those frameworks are not feasible for game development, although they could virtually achieve the same results of our work. At the end of that paragraph we explain how these techniques have a too high overhead that would be problematic for a game.**

the last paragraph mentions meta-programming approaches. Please recall what this is in a sentence and add references. BTW, this last paragraph would be better moved to the Conclusion section, since it mentions future work.

**We moved the entire part on the meta-compiler in the Conclusion as suggested and there we added an explanation on what a meta-compiler is.**

the final paragraph also applies to Casanova 2 ;) Why will your project differ from the others?

**Added an explanation on how Casanova is different from other DSL’s**

recall the meaning of the SOLID acronym, and add a reference to this notion.

**Added explanation and reference**

it would be worthwhile to mention the notion of memoization, since this is a variant of this optimization that is suggested in Section 4.1.

**The proposed optimization is more an indexing optimization (spatial indexing) rather than memoization, because we do not cache the result of any function call, we only index the fleets satisfying the required condition.**

the sentence "predicates belong to" is vague. What is exactly meant by this notion of belonging?

**Rephrased with “defined on” and restructured the sentence for better clarity**

At the end of Section 4.3, the approach based on meta-programming that is suggested there seems overly ambitious. A first step that seems more reachable would be to perform inference on Casanova 2 source code to try to infer the dependencies used by the compiler to implement its optimizations and which are currently manually provided (mentioned in Section 5).

**Removed. Indeed it seems premature, given the state of the meta-compiler itself, to claim that it is possible to extend Casanova itself to include the features of a meta-compiler. In the Conclusion we indeed refer to this as a future work that will be implemented separately in the meta-compiler.**

t would be interesting to provide some examples of the manual annotations required to implement the optimizations you suggest.

**There was a missing “could” that could create the misunderstanding that the feature is actually implemented in Casanova 2 compiler. That was a proposal to further improve the current architecture but it is not yet implemented in our work. We added this in the future works as well.**

I don't recall seeing any mention of the algorithms Atomic(p) and BuildICDS() in the main text. Please reference and describe them, if this is really useful.

**See page 15 in the revised version, both are referenced.**

the notion of "shape of the IC" is not clear; I assume you mean "structure".  Also, in the example, "wait 10" is unclear; what does 10 stand for?

**Changed “shape” into “structure” as suggested. Explained what wait 10 means (wait for 10 seconds) when describing the code.**

I didn't quite understand the last sentence of the "On update" paragraph. Please rephrase.

**Rephrased into: “We store a collection in the world entity that contains all the suspended rules that currently active. These rules are grouped by their respective entities.”**

I assume that "add and remove" is "add" only.

**Fixed and rephrased for better understanding**

in the "Connecting" paragraph, the first sentence uses "once when the player joins the game", while the "Connected" paragraph mentions "whenever a player joins the game". This seems redundant, and, by the way, the two code excerpts are the same.

**Rephrased the sentences and added an extra paragraph to further clarify the difference. It is true that, as it was, it looked redundant but it is not.**

the code excerpt at the end of Paragraph 1 contains the "@" sign; its semantics is not specified. Also, at the beginning of Paragraph 3, it is not clear why there is a "wait not IsKeyDown" instruction; please explain.

**Added explanation on @ and the reason behind the wait statement**

References  
- Please improve your references (e.g., 21 and 47 are not informative enough).

**Improved**

The example at the end of Section 5.1 is reminiscent of flowgraph-based languages or even Petri nets. It would be interesting to contrast your approach with this prior work.

**Since none of us is an expert in either flowgraph-based languages or Petri nets, we felt we couldn't do justice to such a comparison at this time. So we decided to leave such a comparison out of the paper for now.**

In Section 7.3.3, the issue of operation atomicity constraints could be discussed.  In general, there is no mention of implementation issues in this section; does it mean there is nothing new to report about this matter?

**Added a discussion about reliability of transmissions in multiplayer games and explained that this is an acceptable tradeoff for network performance.**

**Reviewer 2**

Your work targets, among others, game developers working for smaller companies where development budgets are smaller. One can wonder if these developers would be able to tackle functional programming, i.e. a less mainstream paradigm than imperative programming (C, C++, Java, ...). I guess some would, while others wouldn't.

**M. Abbadi (co-author of this paper) performed some usability tests with different kinds of users, including students of a course in game development. This study showed that switching Casanova does not offer much of a challenge even for experienced programmers in C++. We did not include these results in this article because we felt it was out of scope, but they are being used in his PhD thesis to show the usability of Casanova.**

Another question concerns the choice of building a system on top of the .NET platform. I imagine it's not such a problem at this stage, and the experimental results already obtained demonstrate the validity of your approach. In order to move forward to production stage, though, it might prove necessary to switch to a more classical platform, e.g. C/C++ under Unix. This should not be very difficult on the principle, but would likely require a complete rewrite, as the Casanova compiler is implemented using F#, a .NET-specific language.

**The choice of F# has been made because, according to our experience, developing a compiler with a functional programming language is generally faster than with an imperative language. Your remark is, of course, true, but I would like to point out that now .NET is a multi-platform framework, being able to run both on MAC-OS and Linux, so its use is no more restricted to Windows only.**