Making games with Casanova

Mohamed Abbadi, Francesco Di Giacomo, Agostino Cortesi, Pieter Spronck, Giulia Costantini, Giuseppe Maggiore
{mohamed.abbadi, francesco.digiacomo, cortesi}@unive.it, p.spronck@uvt.nl, costg@hr.nl, maggg@hr.nl

Introduction

Managing the flow of time and the coordination of multiple components in interactive applications (such as games) is a complex activity. For this reason, game development requires a lot of effort, even for (apparently) simple scenarios. We introduce a new language (Casanova 2), aimed at reducing cost and effort in the the development of interactive applications (through the use of specific constructs), while at the same time achieving high performance. The results obtained on a series of case studies are satisfactory with respect to this goal.

Conclusion

Casanova 2 may offer a solution for the high development costs of games. The goal of Casanova 2 is to reduce the effort and complexities associated with building games. This is achieved through a language designed around games: Casanova 2 manages the game world through entities and rules, and offers constructs (wait and yield) to deal with the run-time dynamics. As shown by our results, we believe that we have taken a significant step towards reaching these goals. In fact, we achieved at the same time very good performance and ease of use, thereby empowering developers with limited resources.

References

[1] Giuseppe Maggiore, Alvise Spanò, Renzo Orsini, Giulia Costantini, Michele Bugliesi, and Mohamed Abbadi. Designing casanova: a language for games. In *Advances in Computer Games*, pages 320–332. Springer, 2012.

[2] Giuseppe Maggiore, Pieter Spronck, Renzo Orsini, Michele Bugliesi, Enrico Steffinlongo, and Mohamed Abbadi. Writing real-time. net games in casanova. In *Entertainment Computing-ICEC 2012*, pages 341–348. Springer, 2012.

[3] Giuseppe Maggiore, Alvise Spanò, Renzo Orsini, Michele Bugliesi, Mohamed Abbadi, and Enrico Steffinlongo. A formal specification for casanova, a language for computer games. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 287–292. ACM, 2012.

[4] Giuseppe Maggiore, Renzo Orsini, and Michele Bugliesi. On casanova and databases or the similarity between games and dbs. In *SEBD*, pages 271–276, 2012.

[5] Mohamed Abbadi, Francesco Di Giacomo, Renzo Orsini, Aske Plaat, Pieter Spronck, and Giuseppe Maggiore. Resources, entities, actions. a generalized design pattern for rts games and its language extension in casanova.

Acknowledgements

We would like to thank Professors Aske Plaat, Fons Maes and Renzo Orsini for their support and contribution.

Past

Figure 1: Some previous Casanova games (left); Galaxy Wars (right); Casanova implementation with F# syntax

Present

Figure 2: Ambulance game for pediatrics (left); Game for dyslexia detection (center); Real-time strategy game(right)

```
world Patrol = {
   V : Vector2
   P : Vector2
   Checkpoints : [Vector2]

   rule P = P + V * dt
   rule V =
    for checkpoint in Checkpoints do
      yield ||checkpoint - P||
      wait P = checkpoint
      yield Vector2.Zero
      wait 10<s>
}
```

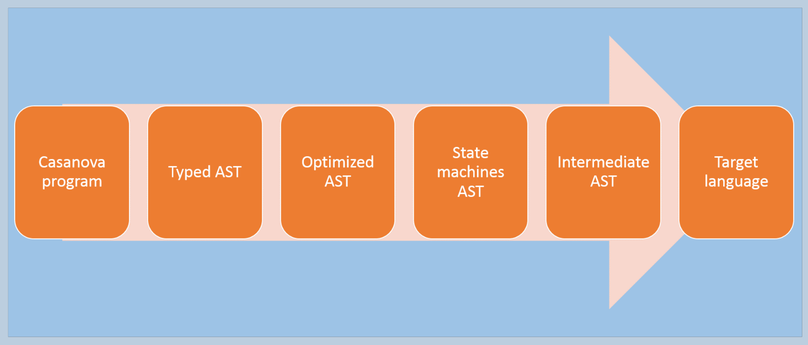

Figure 3: Casanova patrol sample (left); Casanova layered compiler (right)

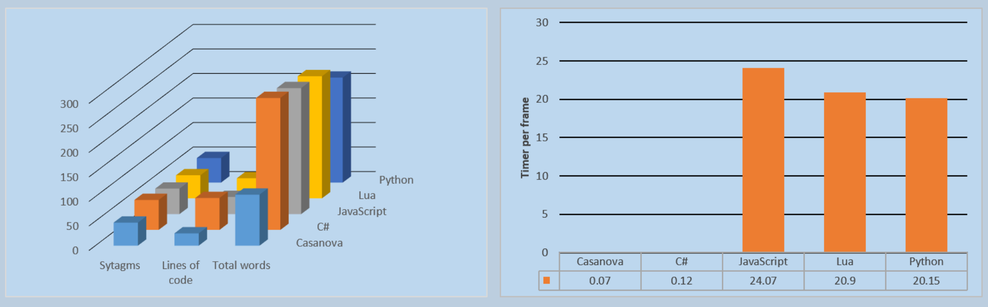

Figure 4: Casanova syntax evaluation (left); Casanova performance evaluation (right)

Future - Casanova logic system

```
winningPlayer {self opponent} usedCells lines ⇒ None
∀ freeCell ∈ freeCells
random 1 100 > threshold
let freeCells' = freeCells \ {freeCell}
let usedCells' = usedCells[freeCell → currentPlayer]
randomWalks self opponent nextPlayer currentPlayer (usedCells',freeCells') lines (threshold - 5) ⇓_{; nil} scores
sum scores ⇒ score
──────────────────────────────────────────────────────────────────────────────
randomWalks self opponent currentPlayer nextPlayer (usedCells,freeCells) lines threshold ⇒ score ∎

winningPlayer {self opponent} usedCells lines ⇒ winner
winner ≡ self
──────────────────────────────────────────────────────────────────────────────
randomWalks self opponent currentPlayer nextPlayer (usedCells,freeCells) lines threshold ⇒ 5 ∎

winningPlayer {self opponent} usedCells lines ⇒ winner
winner ≡ opponent
──────────────────────────────────────────────────────────────────────────────
randomWalks self opponent currentPlayer nextPlayer (usedCells,freeCells) lines threshold ⇒ -10 ∎


winningPlayer {self opponent} usedCells lines ⇒ None
──────────────────────────────────────────────────────────────────────────────
randomWalks self opponent currentPlayer nextPlayer (usedCells,{}) lines threshold ⇒ 2 ∎
```

```
t, ΓLocal, Γ ⇒ t : bool :: unit
u, ΓLocal, Γ ⇒ u : T :: U     v, ΓLocal, Γ ⇒ v : T :: V
U ⊔ V ⇒ T'
──────────────────────────────────────────────────────────
if t then u else v, ΓLocal, Γ ⇒ if t then u else v : T :: T' ∎


t, ΓLocal, Γ ⇒ t : T :: unit     u, ΓLocal[$x↦T], Γ ⇒ u : U :: V
──────────────────────────────────────────────────────────
let $x = t in u, ΓLocal, Γ ⇒ let $x = t in u : U :: V ∎


t, ΓLocal, Γ ⇒ t : T :: unit     u, ΓLocal, Γ ⇒ u : T :: unit
──────────────────────────────────────────────────────────
t * u, ΓLocal, Γ ⇒ t * u : T :: unit ∎
```

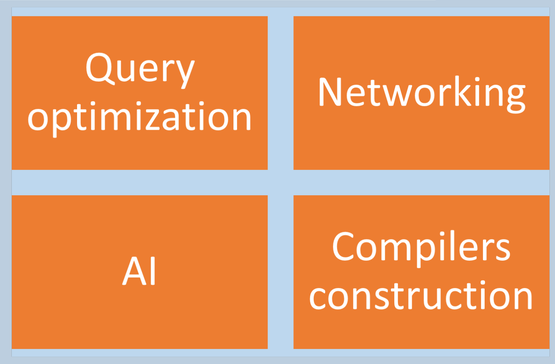

Figure 5: Random walk (left); Type checker (center); Future applications