Casanova: a language for game development

Candidate: G. Maggiore [1,2]
Supervisors: M. Bugliesi [1] P. Spronck [3]

[1]Università Ca' Foscari - Venezia, Italy

[2]NHTV University - Breda, Netherlands

[3]Tilburg University, Netherlands

Table of contents

Outline

Motivation

Game development

- Very large industry
- Impact on
  - technology
  - culture
  - society

Motivation

Challenges in game development

- **Costs and complexity**...

Motivation

Challenges in game development

- **Costs and complexity**...
- ...resulting in less innovation
- Especially by **smaller developers**
  - independent (indie)
  - research
  - serious

Outline

Making a game

Structure of a game

- **game world/state**
- **game loop**
  - continuous logic of the game world (`P := P + V * dt`)
  - drawing of all drawables $\forall$ `d : Drawable` $\in$ `world do draw(d)`
  - discrete logic of the game world (`in_room(player) -> light := ON`)
  - **high performance**

Structure of a game

Game world

```
class World {
  public List<Ship> Ships;
  public List<Planet> Planets;
  ...
}

class Ship {
  public Vector3 Position;
  public Vector3 Velocity;
  ...
  public Sprite HealthBar;
  public Model3D Appearance;
}

...
```

Structure of a game

Game loop

```
void Update(World world, float dt) {
  foreach(var s in world.Ships)
    if (!UpdateShip(world, s, dt))
      world.Ships.Remove(s);
  world.Ships.Add(NewShips(world, dt));
  ...
}
```

Structure of a game

Continuous dynamics

```
void UpdateShip(World w, Ship s, float dt) {
  s.Position += s.Velocity * dt;
  s.Velocity += s.Acceleration * dt;
  s.Life -= s.Damage(w) * dt;
  ...
  return s.Life > 0.0f;
}
```

Structure of a game

Drawing

```
void Draw(World world, float dt) {
  foreach(var s in world.Ships)
    DrawShip(s);
  ...
}

void DrawShip(s) {
  s.HealthBar.Transform =
    Matrix.CreateTranslation(s.Position) *
    Matrix.CreateScale(s.Life, 1.0f, 1.0f);
  DrawSprite(s.HealthBar);
  s.Appearance.Transform =
    Matrix.CreateTranslation(s.Position) *
    Matrix.CreateRotationY(atan2(norm(s.Velocity)));
  DrawModel3D(s.Appearance);
}
```

Structure of a game

Discrete dynamics (state machines)

```
class ShipSpawnTimer {
  public float Time;
  public bool Tick(float dt) {
    Time -= dt;
    if (Time <= 0.0f) {
      Time = ShipSpawnTime;
      return true;
    } else
      return false;
  }
}

Seq<Ship> NewShips(World world, float dt) {
  foreach(var s in world.ShipSpawnTimers)
    if(s.Tick(dt)) yield new Ship(...);
}
```

Structure of a game

Discrete dynamics (state machines)

```
class DragSelector {
  DragSelectionState state;
  Vector2 start;
  public Rectangle? Tick() {
    switch(state)
      case NotStarted:
        if (Mouse.LeftButton == ButtonState.Down) {
          state = Started;
          start = Mouse.Position;
          CreateSelectionRectangle(start);
          return null; }
      case Started:
        if (Mouse.LeftButton == ButtonState.Up) {
          state = Ended;
          end = Mouse.Position;
          return null; }
      case Ended:
        state = NotStarted;
        RemoveSelectionRectangle();
        return new Rectangle(start, end);
  }
}
```

Structure of a game

Optimization

```
class Ship {
  public float Damage(World w) {
    // use a spatial partitioning index to quickly find the
    // adversary's ships close enough to this one to damage it
    ...
  }
```

Game code

The code above is:

- **a lot:** traversing, updating, and drawing a large and complex game world
- **error-prone:** explicit handling of state machines, keeping in sync update and draw code
- **complex:** order of updates, order of draws, optimization algorithms
- **boilerplate:** large portion of code not specific to the game

Game code

Coping with complexity

- **libraries**
  - low-level libraries of utilities
  - high-level OO libraries for game worlds (components)
- **game engines**
  - scripting for customizing an existing game
  - visual environment to fill the scene graph

Outline

Casanova

A game-development DSL

- **part of the ML family**
- **specific syntax and semantics**
  - less code
  - less complexity
  - less boilerplate
- **specific optimizations**
  - higher performance
  - less complexity
  - less bugs

Casanova

Features

- **no game loop**
- update logic through declarative, referentially transparent, local **rules**
- **automated, declarative drawing** with transparent batching
- state machines through (a calculus of) **coroutines**
- **automated optimizations**

Casanova

Syntax - ML Types

```
<type-decls> ::= <type-decl> | <type-decl> <type-decls>
<type-decl>  ::= 'type' <id> '=' <type-body>
<type-body>  ::= <record-body> | <union-body>

<type-expr>    ::= <id> | <tuple-type-expr> | <intrinsic-type-expr>
<tuple-type-expr> ::= <type-expr> | <type-expr> '*' <tuple-type-expr>
<intrinsic-type-expr> ::= 'Var<'<type-expr>'>'
                        | 'Ref<'<type-expr>'>'
                        | 'List<'<type-expr>'>'
                        | 'Coroutine<'<type-expr>'>'
                        | float | int | Vector2 | ...
<union-body>  ::= <id> | <id> 'of' <type-expr>
                | <id> '|' <union-body>
                | <id> 'of' <type-expr> '|' <union-body>
```

Casanova

Syntax - Rules

```
<record-body> ::= '{' <labels> '}' <rules>
<labels>      ::= <label> | <label>';' <labels>
<label>       ::= <id> ':' <type-expr>
<rules>       ::= empty | <rule> <rules>
<rule>        ::= 'rule' <rule-id> '=' <expr>
<rule-id>     ::= <id> | <id>'.'<rule-id>
```

Casanova

Syntax - ML expressions

```
<type-init>     ::= <tuple-init> | <list-init>
                  | <record-init> | <union-init> | <intrinsic-type-init>
<tuple-init>    ::= <expr> | <expr>, <tuple-init>
<list-init>     ::= '[' <expr-list> ']' | '[' <list-compr> ']'
<list-compr>    ::= 'for' <id> 'in' <expr> 'do' <expr>
<expr-list>     ::= '' | <expr> ';' <expr-list>
<record-init>   ::= '{' <labels-init> '}'
<labels-init>   ::= <id> '=' <expr> | <id> '=' <expr> ';' <labels-init>
<union-init>    ::= <id> | <id> <tuple-init>
<intrinsic-type-init> ::= 'ref' <expr> | 'var' <expr>
                  | 'Vector2(' <expr> ',' <expr> ')' | ...
<type-dest>     ::= 'let' <ids> '=' <expr> | <match-case>
                    | '[]' | <id> '::' <id> | <id> '.' <id>
<match-case>    ::= 'match' <expr> 'with' <patterns>
<patterns>      ::= <pattern> | <pattern> '|' <patterns>
<pattern>       ::= <id> | <ids> | <id> '(' <pattern-args> ')'
<pattern-args> ::= <pattern-arg> | <pattern-arg> ',' <pattern-args>
<pattern-arg>   ::= <id> | <const>
<id-decl> ::= <id> | <id> ':' <type-expr>
<ids>           ::= <id> | <id> ',' <ids>
<id>       ::= ... (* an alphanumeric string *)
<const>    ::= ... (* a constant value *)
```

Casanova

Syntax - ML expressions/coroutines

```
<expr>     ::= <id> | <const>
             | 'let' <id> '=' <expr> 'in' <expr>
             | 'if' <expr> 'then' <expr> 'else' <expr>
             | <type-init> | <type-dest>
             | <co-expr> | ...
<co-expr> ::= 'co{' <co-expr> '}' | <expr> | 'return' <expr>
             | 'let!' <id> '=' <expr> 'in' <expr>
             | 'do!' <expr> ';' <expr>
             | <expr> '||' <expr> | <expr> '&&' <expr>
             | <expr> '=>' <expr> | 'repeat' <expr>
             | 'yield'
```

Casanova

Syntax - Program

```
<p>               ::= <type-decls> <initial-world> <scripts>
<initial-world> ::= 'let world =' <expr>
<scripts>         ::= <main-script> <input-script>
<main-script>    ::= 'let main =' <expr>
<input-script>   ::= 'let input =' <expr>
```

Casanova

Typing rules

```
type E = { l1:T1; l2:T2; ... ln:Tn }
  rule li = (ei:World * E * float<s> -> Ti)
```

Casanova

Typing rules

```
type E = { l1:T1; ...; li:{k1:V1; k2:V2; ... km:Vm} ln:Tn }
  rule li.kj = (eij:World * E * float<s> -> Vj)
```

Casanova

ML-style typing rules

$$\frac{\Gamma \vdash t_1 : \mathtt{U},\ \Gamma, x : \mathtt{U} \vdash t_2 : \mathtt{V}}{\Gamma \vdash \mathtt{let}\ x{=}t_1\ \mathtt{in}\ t_2 : V} \qquad \frac{\Gamma \vdash cond : \mathtt{bool},\ t_1 : \mathtt{U},\ t_2 : \mathtt{U}}{\Gamma \vdash \mathtt{if}\ cond\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 : U} \qquad \frac{\Gamma \vdash f : U \rightarrow V, t : U}{\Gamma \vdash ft : V}$$

$$\frac{\Gamma \vdash t : \{l_1 : T_1; l_2 : T_2; \ldots l_n : T_n\}}{\Gamma \vdash t.l_i : T_i} \qquad \ldots$$

Casanova

Coroutines typing rules

$$\frac{\Gamma \vdash t_1 : \texttt{Var<T>}}{\Gamma \vdash !t_1 : \texttt{T}} \qquad \frac{\Gamma \vdash t_1 : \texttt{Var<T>}, t_2 : \texttt{T}}{\Gamma \vdash t_1 := t_2 : \texttt{Co<Unit>}} \qquad \frac{\Gamma \vdash x : \texttt{T}}{\Gamma \vdash \texttt{return}\ x : \texttt{Co<T>}}$$

$$\frac{\Gamma \vdash t1 : \texttt{Co<U>}\Gamma, x : \texttt{U} \vdash t2 : \texttt{Co<V>}}{\Gamma \vdash \texttt{let!}\ \ x\texttt{=}t1\ \texttt{in}\ t2 : \texttt{Co<V>}} \qquad \vdash \texttt{yield} : \texttt{Co<Unit>} \qquad \frac{\Gamma \vdash t : \texttt{float<s>}}{\Gamma \vdash \texttt{wait}\ t : \texttt{Co<Unit>}}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Co<U>}, t_2 : \texttt{Co<V>}}{\Gamma \vdash t_1\ \texttt{\&\&}\ t_2 : \texttt{Co<U*V>}} \qquad \frac{\Gamma \vdash t_1 : \texttt{Co<U>}, t_2 : \texttt{Co<V>}}{\Gamma \vdash t_1\ \texttt{||}\ t_2 : \texttt{Co<Either<U,V>>}} \qquad \frac{\Gamma \vdash t_1 : \texttt{Co<bool>}, t_2 : \texttt{Co<V>}}{\Gamma \vdash t_1\ \texttt{=>}\ t_2 : \texttt{Co<V>}}$$

$$\frac{\Gamma \vdash t : \texttt{Co<Unit>}}{\Gamma \vdash \texttt{repeat}\ t : \texttt{Co<Unit>}}$$

Casanova

Overall game semantics

```
let rec update world dt =
  let world'  = apply_rules world dt
  let world'' = tick_scripts world'
  do draw world''
  update world'' dt
```

Casanova

World traversal semantics

```
apply_rules (world:World) (dt:float<s>) = update_world [World] world dt

update_world [World] (world:World) (dt:float<s>) =
  update_entity [World] [World] world world dt

update_entity [World] [Primitive(T)]
              (world:World) (v:T) dt = ()
update_entity [World] [T1 * T2 * ... * Tn]
              (world:World) (x1:T1, x2:T2, ..., xn:Tn) dt =
  update_entity [World] [T1] world x1 dt
  update_entity [World] [T2] world x2 dt
  ...
  update_entity [World] [Tn] world xn dt
update_entity [World] [Var<T>]
              (world:World) (v:Var<T>) dt =
  update_entity [World] [T] (world:World) !v dt
update_entity [World] [Ref<T>]
              (world:World) (v:Ref<T>) dt = ()
update_entity [World] [List<T>]
              (world:World) (l:List<T>) dt =
  for x in l do update_entity [World] [T] world x dt
```

Casanova

World traversal semantics

```
update_entity [World] [T=UnionCase(C1(T11 * ... * T1n1),
                                   C2(T21 * ... * T2n2), ...,
                                   Cn(Tn1 * ... * Tnnn))]
              (world:World) (c:T) dt =
  match c with
  | C1(x1,...,xn1) ->
    update_entity [World] [T11] world x1 dt
    update_entity [World] [T12] world x2 dt
    ...
    update_entity [World] [T1n1] world xn1 dt
  | C2(x1,...,xn2) ->
    update_entity [World] [T21] world x1 dt
    update_entity [World] [T22] world x2 dt
    ...
    update_entity [World] [T2n1] world xn2 dt
  ...
  | Cn(x1,...,xnn) ->
    update_entity [World] [Tn1] world x1 dt
    update_entity [World] [Tn2] world x2 dt
    ...
    update_entity [World] [Tnnn] world xnn dt
```

Casanova

Rule application semantics

```
update_entity [World] [T=Record(l1:T1,l2:T2,...,ln:Tn,
                                r1=rb1,r2=rb2,...,rm=rbm)]
              (world:World) (r:T) dt =
  update_entity [World] [T1] world r.l1 dt
  update_entity [World] [T2] world r.l2 dt
  ...
  update_entity [World] [Tn] world r.ln dt
  r.r1 := rb1(world, r, dt)
  r.r2 := rb2(world, r, dt)
  ...
  r.rm := rbm(world, r, dt)
```

Casanova

Coroutines semantics

```
type ScriptStep<s,a> = Done of a * s | Next of Script<s,a> * s
and Script<s,a> = s → ScriptStep<s,a>

let s1 >>= s2 =
  fun s ->
    match s1 s with
    | Done(x,s') -> Next(s2 x,s')
    | Next(k,s') -> Next(k >>= s2,s')

let return x = fun s -> Done x

let yield = fun s -> Right(s, (fun s -> ((),s)))
```

Casanova

Coroutines operators semantics

```
let (&&) (p:Co<'a>) (q:Co<'b>) : Co<'a * 'b> =
  match p(), q() with
  | Done x, Done y -> Done(x,y)
  | Next p', Next q' -> Next (p' && q')
  | Next p', Done y -> Next(p' && return y)
  | Done x, Next q' -> Next(return x && q')

let (||) (p:Co<'a>) (q:Co<'b>) : Co<Choice<'a,'b>> =
  match p(), q() with
  | Done x, _ -> Done(Choice1Of2 x)
  | _, Done y -> Done(Choice2Of2 y)
  | Next p', Next q' -> Next (p' || q')

let repeat (p:Co<Unit>) : Co<Unit> = p >>= (fun _ -> repeat p)
```

Casanova

Draw batching semantics

```
layers = []
draw_world [World] (world:World) =
  draw_entity [World] world
  for layer in layers do
    layer.Draw()
    layer.Clear()
```

Casanova

Draw traversal semantics

```
draw_entity [Primitive(T)] (v:T) = ()
draw_entity [T1 * T2 * ... * Tn] (x1:T1, x2:T2, ..., xn:Tn) =
  draw_entity [T1] x1
  ...
  draw_entity [Tn] xn
draw_entity [Var<T>] (v:Var<T>) = draw_entity [T] !v
draw_entity [Ref<T>] (v:Ref<T>) = ()
draw_entity [List<T>] (l:List<T>) =
  for x in l do draw_entity [T] x
```

Casanova

Draw traversal semantics

```
draw_entity [T=UnionCase(C1(T11 * ... * T1n1),
                         Cn(Tn1 * ... * Tnm)))] (c:T) =
  match dt with
  | C1(x1,...,xn1) ->
    draw_entity [T11] x1
    ...
    draw_entity [T1n1] xn1
  ...
  | Cn(x1,...,xnm) ->
    draw_entity [Tn1] x1
    ...
    draw_entity [Tnm] xnm

draw_entity [T=Record(l1:T1,l2:T2,...,ln:Tn,rules)] (r:T) =
  draw_entity [T1] r.l1
  ...
  draw_entity [Tn] r.ln
```

Casanova

Actual drawing semantics

```
draw_entity [Drawable(T)] (d:T) = d.Layer.AddT(d)
draw_entity [Layer(T)] (l:T) = layers.Add(l)
```

Note: easily extensible with additional drawable datatypes.

Structure of a game

Game of life

```
type World = {
  Cells : List<Cell>
  UpdateNow : Var<bool>
}

type Cell = {
  Position : Vector2<m>
  Value    : int
  Sprite   : Sprite }
  rule Value(world,self,dt) =
    if world.UpdateNow then
      let around = sum [for c in world.Cells do
                          if dist(self.Position, c.Position < 1.5f<m> then
                            yield c.Value]
      match around with
      | 3 -> 1
      | 2 -> self.Value
      | _ -> 0
    else self.Value
  rule Sprite.Color(world,self,dt) =
    if self.Value = 0 then Color.Black else Color.White
```

Casanova

Structure of a game

```
let main world =
  repeat co{
    do! wait 1.0<s>
    do! world.UpdateNow := true
    do! yield
    do! world.UpdateNow := false }

let input = [
    wait_key_down Keys.Space =>
     co{
       do! world.UpdateNow := true
       do! yield
       do! world.UpdateNow := false }
  ]
```

Casanova

Implementation

- presented syntax is not implemented in practice
- *embedding* into F#, leveraging existing high quality:
  - development environment
  - debugging tools
  - libraries (MonoGame)
- Casanova compiler
  - compiles a *stub* dll with the F# compiler
  - *reflects* the existing type and script definitions
  - *emits* the missing functionality in the final dll

Casanova

Type translation into F#

```
translate_types [] = ()
translate_types type::types =
  translate_type type
  translate_types types

translate_type Primitive(type) = type
translate_type Var<type> = Var<translate_type type>
translate_type List<type> = List<translate_type type>
translate_type Union(cases) = Union([translate_case case | case in cases])
translate_type Tuple(types) = Tuple([translate_type type | type in types])
translate_type Record(labels,rules) = Record([translate_label label rules |
     label in labels])

translate_case UnionCase(case,types) = UnionCase(case, [translate_type type |
     type in types])
translate_label Label(name,type) rules =
  if exists(rule.Name = name) in rules then
    if type = List<type'> then
      Label(name,RuleList<translate_type type'>)
    else
      Label(name,Rule<translate_type type'>)
```

Casanova

Rule double buffering

```
type Rule<'T> = {
  mutable current : 'T;
  mutable next : 'T }

let (!) r = r.current
let (:=) r v' = r.next <- v'
```

Casanova

Optimizations

- aggressive inlining
- multi-threaded draw/update
- memory recycling across frames
- batched drawing

Outline

Evaluation

Comparison with other languages

	Cnv	C#/XNA	pygame	C++/DX
Game loop	V	V	V	V
State traversal	V	X	X	X
State machines	V	X (iter.)	X (iter.)	X
Drawing	V	X	V	X
High performance	V	V	X	V

Structure of a game

C#

```
class World {
  public List<Ship> Ships;
  public List<Planet> Planets;
  ...
}
```

Casanova

```
type World = {
  Ships    : List<Ship>
  Planets  : List<Planet>
  ...
}
```

Structure of a game

C#

```
void Update(World world, float dt) {
 foreach(var s in world.Ships)
   if (!UpdateShip(world, s, dt))
     world.Ships.Remove(s);
 world.Ships.Add(NewShips(world,dt));
  ...
}
```

Casanova

```
type World = {
  ..
} rule Ships =
   [for s in self.Ships do
     if s.Life > 0.0f then yield s]
```

One less source of bugs for free!

Structure of a game

C#

```
void UpdateShip(World w, Ship s,
     float dt) {
  s.Position += s.Velocity * dt;
  s.Velocity += s.Acceleration * dt;
  s.Life -= s.Damage(w) * dt;
  ...
  return s.Life > 0.0f;
}
```

Casanova

```
type Ship = {
  ...
 } rule Position   = self.Position +
      self.Velocity * dt
   rule Velocity   = self.Velocity +
       self.Acceleration * dt
   rule Life       = self.Life - ...
```

Structure of a game

C#

```
void Draw(World world, float dt) {
  foreach(var s in world.Ships)
    DrawShip(s);
  ...
}

void DrawShip(s) {
  s.HealthBar.Transform =
    Matrix.CreateTranslation(s.
         Position) *
    Matrix.CreateScale(s.Life, 1.0f,
         1.0f);
  DrawSprite(s.HealthBar);
  s.Appearance.Transform =
    Matrix.CreateTranslation(s.
         Position) *
    Matrix.CreateRotationY(atan2(norm
         (s.Velocity)));
  DrawModel3D(s.Appearance);
}
```

Casanova

```
type Ship = {
  ...
  HealthBar  : Line
  Appearance : Model
 } rule HealthBar.Position   = self.
      Position
   rule HealthBar.Length     = self.
        Life
   rule Appearance.Position = self.
        Position
   rule Appearance.Rotation = atan2(
        norm(s.Velocity))
```

Structure of a game

C#

```
class ShipSpawnTimer {
  public float Time;
  public bool Tick(float dt) {
    Time -= dt;
    if (Time <= 0.0f) {
      Time = ShipSpawnTime;
      return true;
    } else
      return false;
  }
}

Seq<Ship> NewShips(World world, float
      dt) {
  foreach(var s in world.
       ShipSpawnTimers)
    if(s.Tick(dt)) yield new Ship
        (...);
}
```

Casanova

```
repeat
 co{
   do! wait ShipSpawnTime
   do! world.Ships.Add(new Ship(...))
 }
```

Structure of a game

C#

```
class DragSelector {
  DragSelectionState state;
  Vector2 start;
  public Rectangle? Tick() {
    switch(state)
      case NotStarted:
        if (Mouse.LeftButton ==
             ButtonState.Down) {
          state = Started;
          start = Mouse.Position;
          CreateSelectionRectangle(
               start);
          return null; }
      case Started:
        if (Mouse.LeftButton ==
             ButtonState.Up) {
          state = Ended;
          end = Mouse.Position;
          return null; }
      case Ended:
        state = NotStarted;
        RemoveSelectionRectangle();
        return new Rectangle(start,
             end); } }
```

Casanova

```
repeat
 co{
   let! start = wait_left_mouse_down
   do CreateSelectionRectangle(start)
   let! end = wait_left_mouse_up
   do RemoveSelectionRectangle()
 }
```

Evaluation

Comparisong with systems

	Cnv	Unity	Game maker
Game loop	V	V	V
State traversal	V	V	V
State machines	V	V (no ser.)	X
Drawing	V	V	V
Comparison	V	V	X
Custom games	V	V (fixed comp.)	X

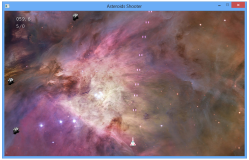
Evaluation

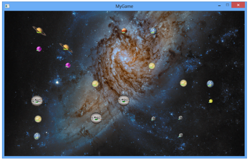
Making games with Casanova?

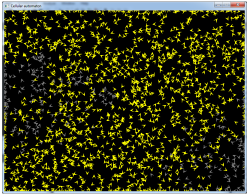
- several samples
- research prototypes
- student games
- even a commercial game!

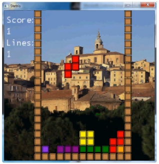
Evaluation

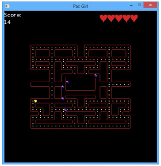
DEMO











Outline

Future work

Already built

- SQL-style, declarative actions to implement resource economies
- declarative menus and global state transitions
- nesting of visuals for relative positioning
- actual assembly, no interpretation steps

About actions

C#

```
class Ship {
  public float Damage(World w) {
    // use a spatial partitioning
         index to quickly find the
    // adversary's ships close enough
          to this one to damage it
    ...
  }
```

Casanova

```
type Ship = {
  ...
  FightAction : TARGET Fleet
                RESTRICTION <>Owner
                     and
                RADIUS < 150.0f
                TRANSFER Life - 1.0
}
```

Future work

Under construction

- networking support
- support for mobile/non-Windows platforms
- support for 3D and hierarchical visibility culling
- integration with Unity 3D
- audio support
- compiler front-end
- static analysis tools and techniques (correct dynamics, load upper bounds)

Conclusions

Impact of Casanova

- game development is **relevant**
- game development is **expensive**
- Casanova makes it **easier, quicker, and safer**
- hopefully to the benefit of smaller developers: **indie, research, and serious**

Conclusions

Relevant publications - preliminary studies

- Giuseppe Maggiore, Michele Bugliesi, and Renzo Orsini. Monadic scripting in F# for computer games. *TTSS, 2011*.
- Giuseppe Maggiore, Fabio Pittarello, Michele Bugliesi, and Mohamed Abbadi. A compilation technique to increase x3d performance and safety. *SAC, 2012*.
- G. Maggiore and G. Costantini. Friendly F# (*book*).

Conclusions

Relevant publications - Casanova

- Giuseppe Maggiore, Alvise Spanò, Renzo Orsini, Giulia Costantini, Michele Bugliesi, and Mohamed Abbadi. Designing casanova: A language for games. *ACG, 2011*.
- Giuseppe Maggiore, Alvise Spanò, Renzo Orsini, Michele Bugliesi, Mohamed Abbadi, and Enrico Steffinlongo. A formal specification for casanova, a language for computer games. *EICS, 2012*.
- Giuseppe Maggiore, Renzo Orsini, and Michele Bugliesi. On casanova and databases or the similarity between games and dbs. *SEBD, 2012*.
- Giuseppe Maggiore, Pieter Spronck, Renzo Orsini, Michele Bugliesi, Enrico Steffinlongo, and Mohamed Abbadi. Writing real-time .Net games in casanova. *ICEC, 2012*.

Conclusions

Relevant publications - work in progress

- LGOAP: adaptive layered planning for real-time videogames. Giuseppe Maggiore, Carlos Santos, Dino Dini, Frank Peters, Hans Bouwknegt, and Pieter Spronck (*submitted to IEEE 2013 Conference on Computational Intelligence in Games*).
- The Domain of Parametric Hypercubes for Static Analysis of Computer Games Software. Giulia Costantini, Pietro Ferrara, Giuseppe Maggiore, and Agostino Cortesi (*submitted to 15th International Conference on Formal Engineering Methods*).
- Resources, Entities, Actions. A generalized design pattern for RTS games and its language extension in Casanova. Mohamed Abbadi, Francesco Di Giacomo, Renzo Orsini, Aske Plaat, Pieter Spronck, and Giuseppe Maggiore. (*submitted to 8TH INTERNATIONAL CONFERENCE ON COMPUTER AND GAMES 2013*).

Conclusions

Relevant publications - our game

- Galaxy Wars. `http://www.galaxywarsthegame.com/`.

This is it

Thank you!

Questions?