UGE in a Nutshell

A Concise, Illustrated Guide to UGE's Game World Simulation, Features and Approaches

Franco Eusébio Garcia (<franco.garcia@dc.ufscar.br>) - Last Update: 04/04/2014

UGE is an open-source game engine for accessible games. It is available at: <https://github.com/francogarcia/uge>

UGE in a Nutshell

UGE is game engine aimed to help prototyping (simple) Universally-Accessible Games (UA-Games).

UA-Games aim to enable users with different interaction (dis)abilities to play*.

UGE is open source and freely available at: https://github.com/francogarcia/uge.

*For more information, please refer to: <http://www.gamasutra.com/view/feature/1764/unified_design_of_universally_.php>.

Introduction

- Creating a (complex) game design accessible for everyone might be impossible.
- Creating a game accessible to different publics via different ways to interact with it is a possibility explored by different papers available in the Literature.
  - For instance, the Unified Design describes an approach to design IO-free game tasks.
  - It is important to note the games should be simple!
    Simpler games such as Pong, Snake and Space Invaders are much more complex to design and implement when we consider many different interaction abilities.
- We know how to implement games to average users.
  What if we could reuse the implementation to make the game accessible for different publics?

Goals

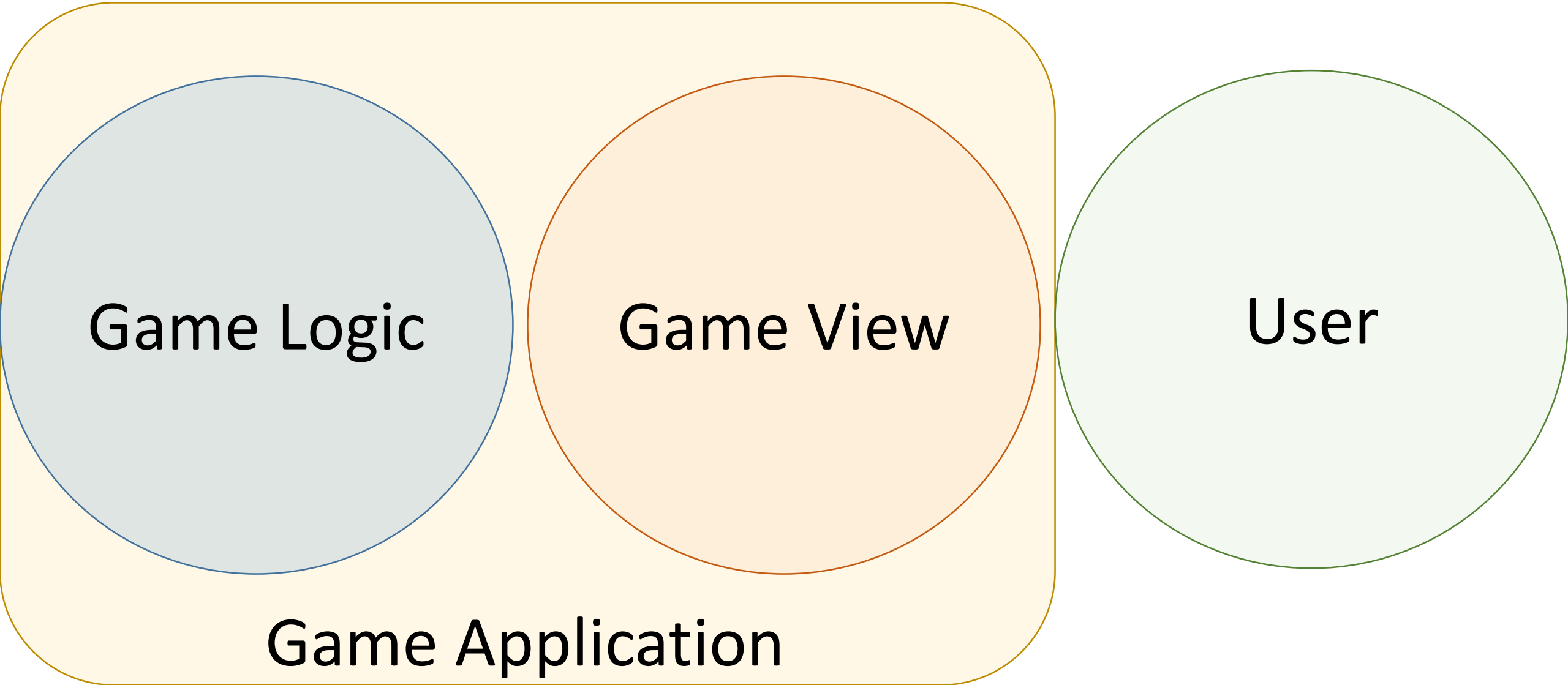
- UGE goals are to:
  - Help games to provide an accessible, enjoyable experience for users with different interactions needs and abilities.
  - Offer extensible, flexible run-time specialization;
  - Ease the creation of configurable, tailor-able game content by:
    - Offering customizable input and output;
    - Ease automation of game input;
    - Allowing users to customize the game to their abilities.
  - Reuse as much as possible Game Logic code;
  - Ease adapting the game for different disabilities and user needs using configurable player profiles.

Goals

- Differently of most game engines, UGE trades performance for run-time flexibility and adaptability.
  - UGE rewrites and extends the very flexible McShaffy and Graham's GCC4 engine* to create a coherent engine for run-time tailoring.
- Trying to achieve the previous goals, the engine combines different architectures:
  - Data-driven;
  - Event-driven;
  - Entity-Component based.
- We explore the data-driven architecture to provide flexible player profiles, which helps to tailor the game without recompiling.

* For more information, please refer to <http://www.mcshaffry.com/GameCode/> and <http://code.google.com/p/gamecode4/>.

UGE Game Elements



UGE Game Elements

Game Logic
(Layer)
An Input-Output
free game simulation

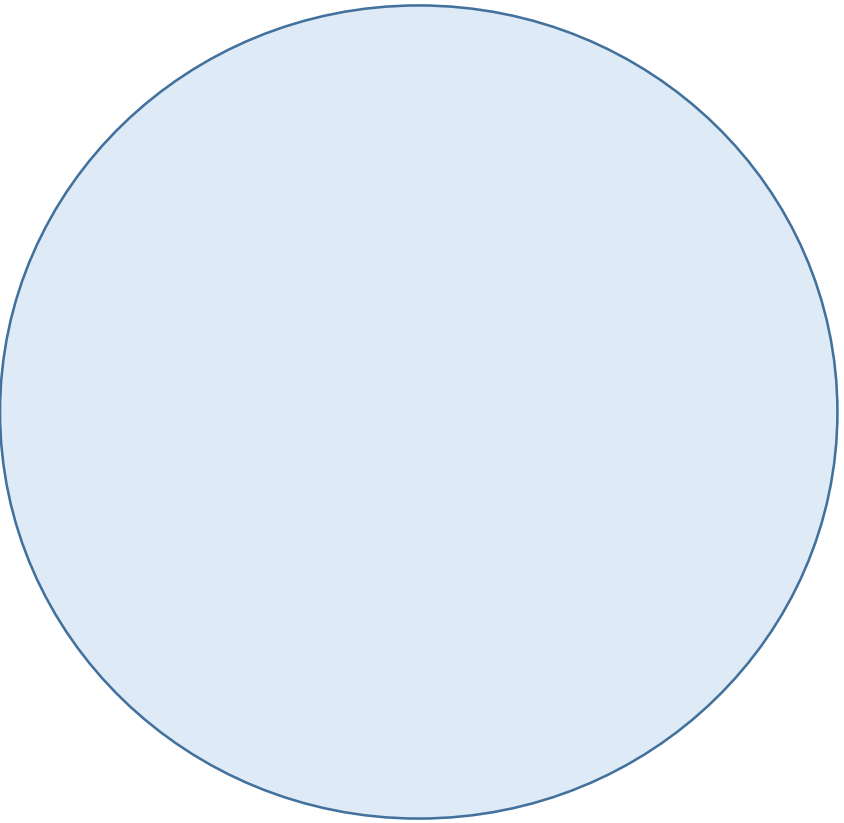
Game View
(Layer)
The game's IO with the
physical level interaction

User
The human who plays
the game

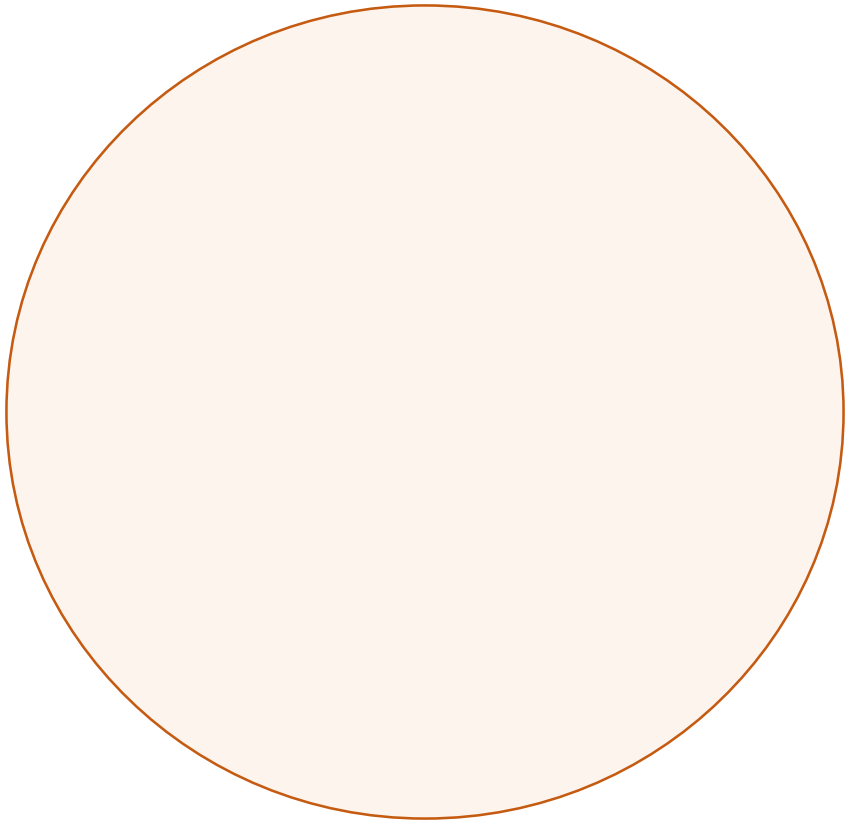
Game Application (Layer)
The game distributed to the user

Game World

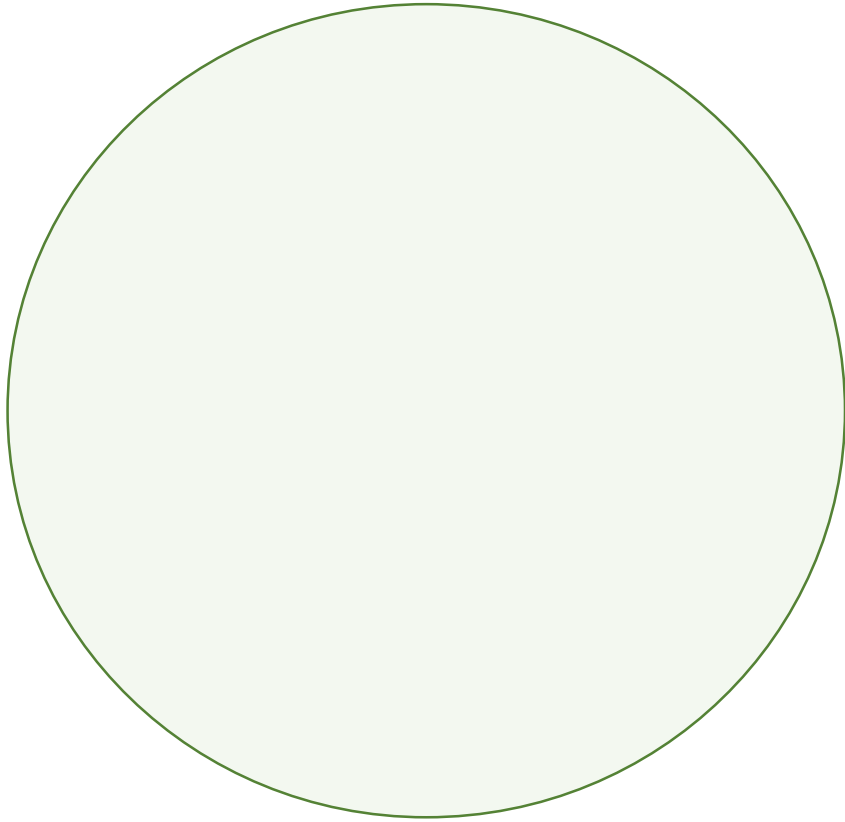
Game Logic



Game View



User

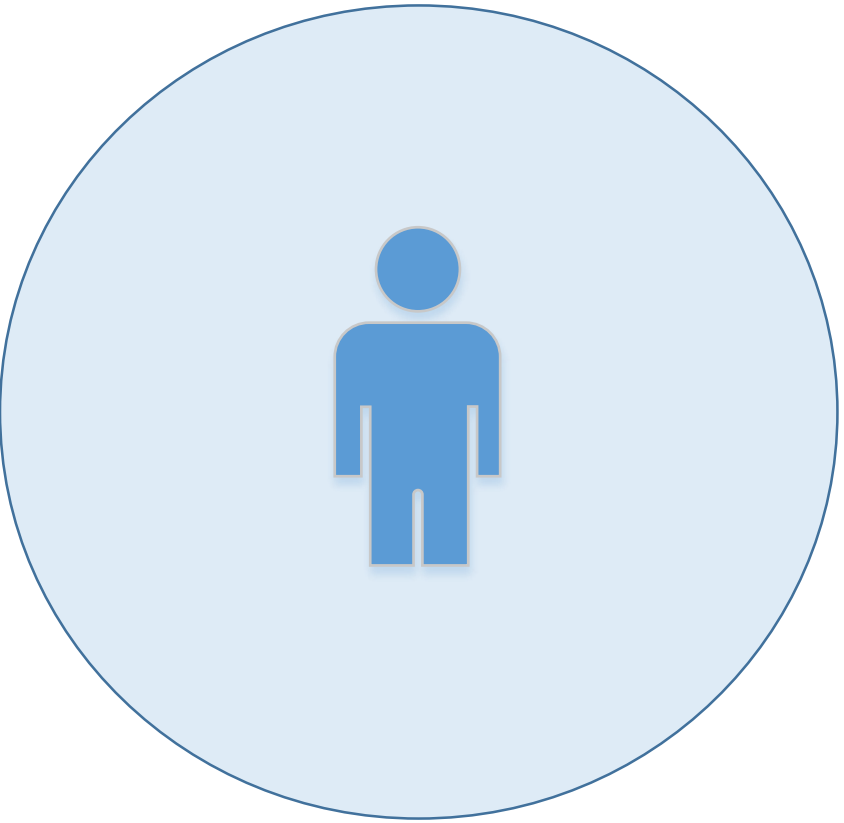


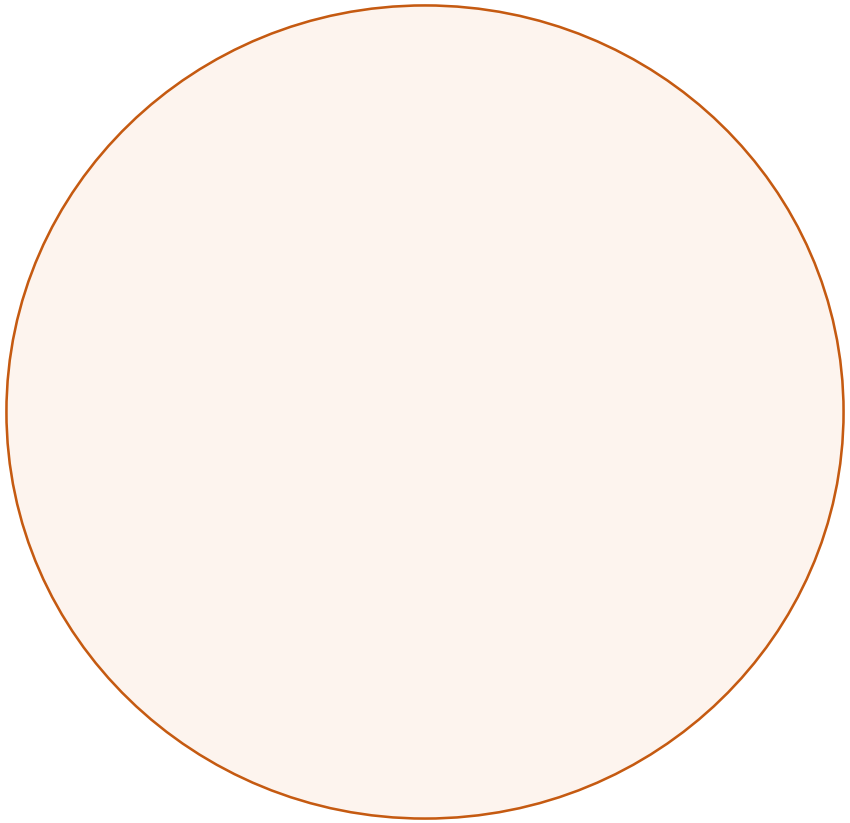
Game World

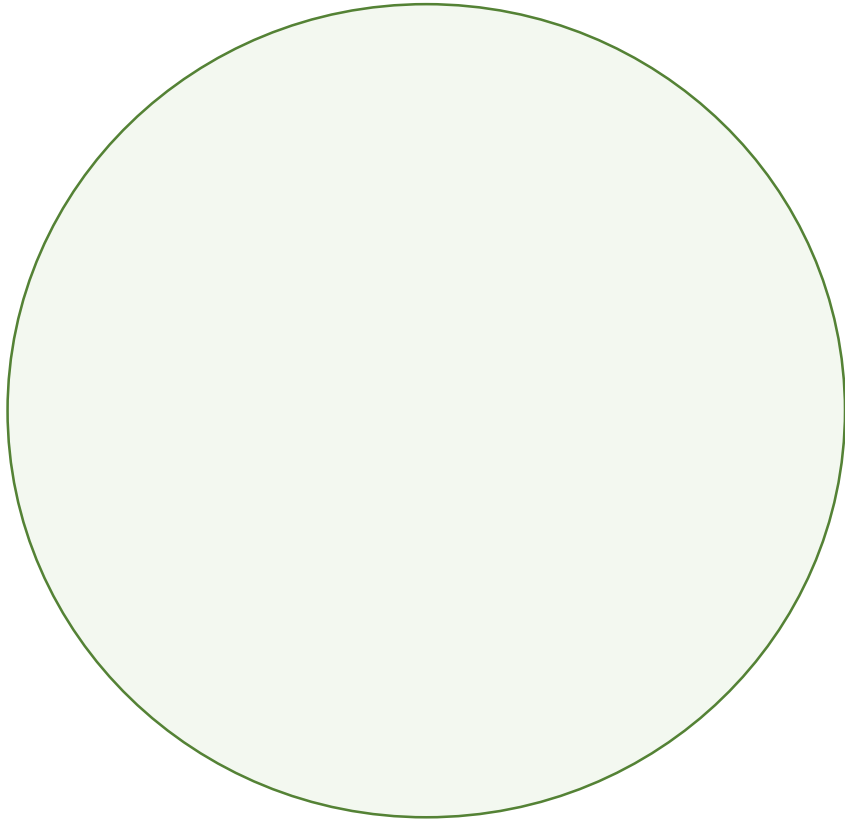
- The game world abstracts UGE's game simulation.
- It starts empty, with no rules or relationships.
- Game developers create the world's rules and behavior.
  UGE simulates the game according to the established definitions.
- In a UGE game, the game world is input-output (IO) free.
  This will have interesting effects later on, allowing IO specializations to the game.

Actor

Game Logic



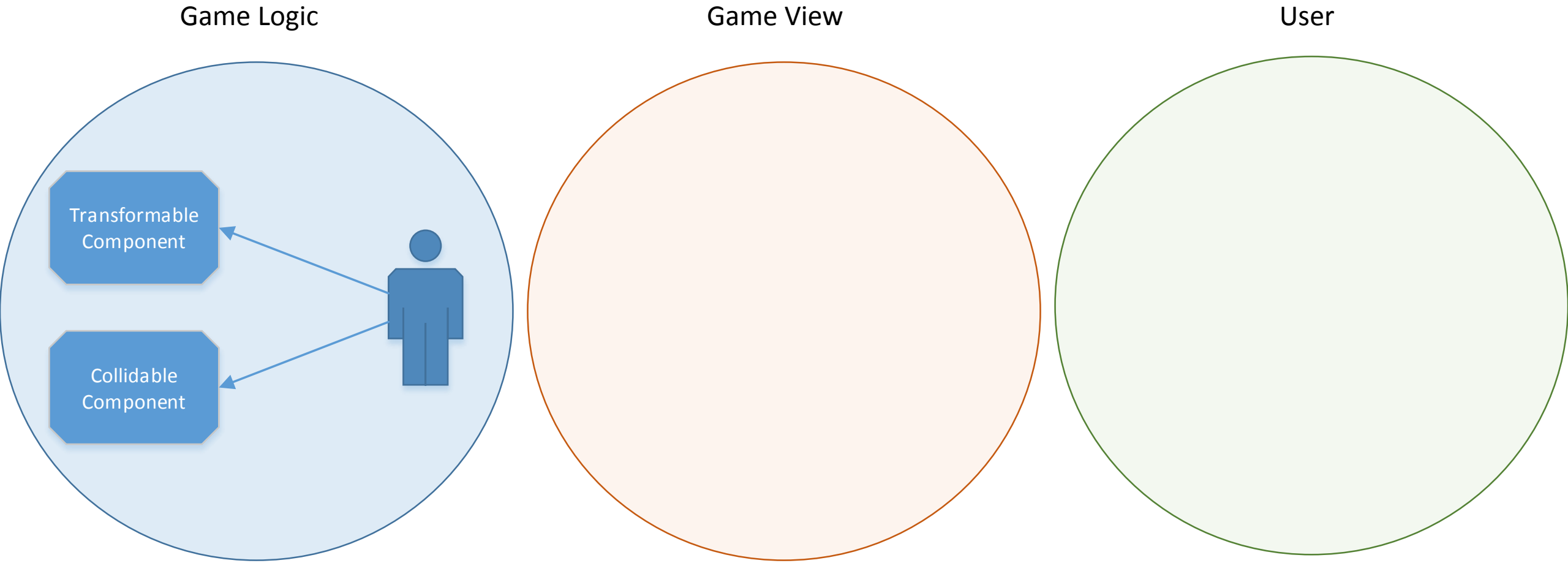




Actor

- The actor is UGE's game entity. An actor is a character, a object or a scenery element that interacts with the game world.
- In the same way an actor play roles to compose, shape and set up a show, game actors shape the game world and create the game experience.
- The UGE's Actor class is straightforward: it has an identifier (`ActorID`). An Actor's complexity comes from a collection of its…

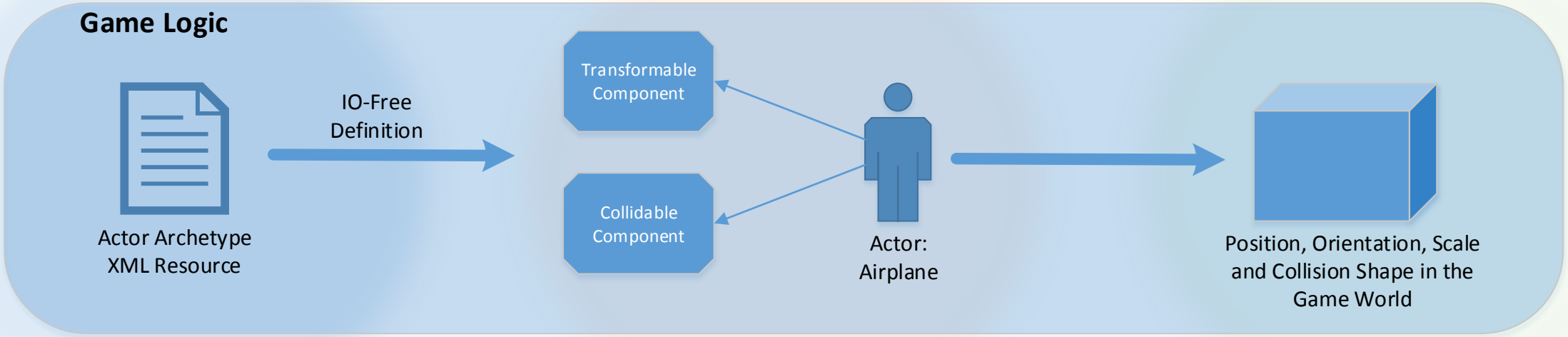
Components


Game Logic
Game View
User
Transformable Component
Collidable Component

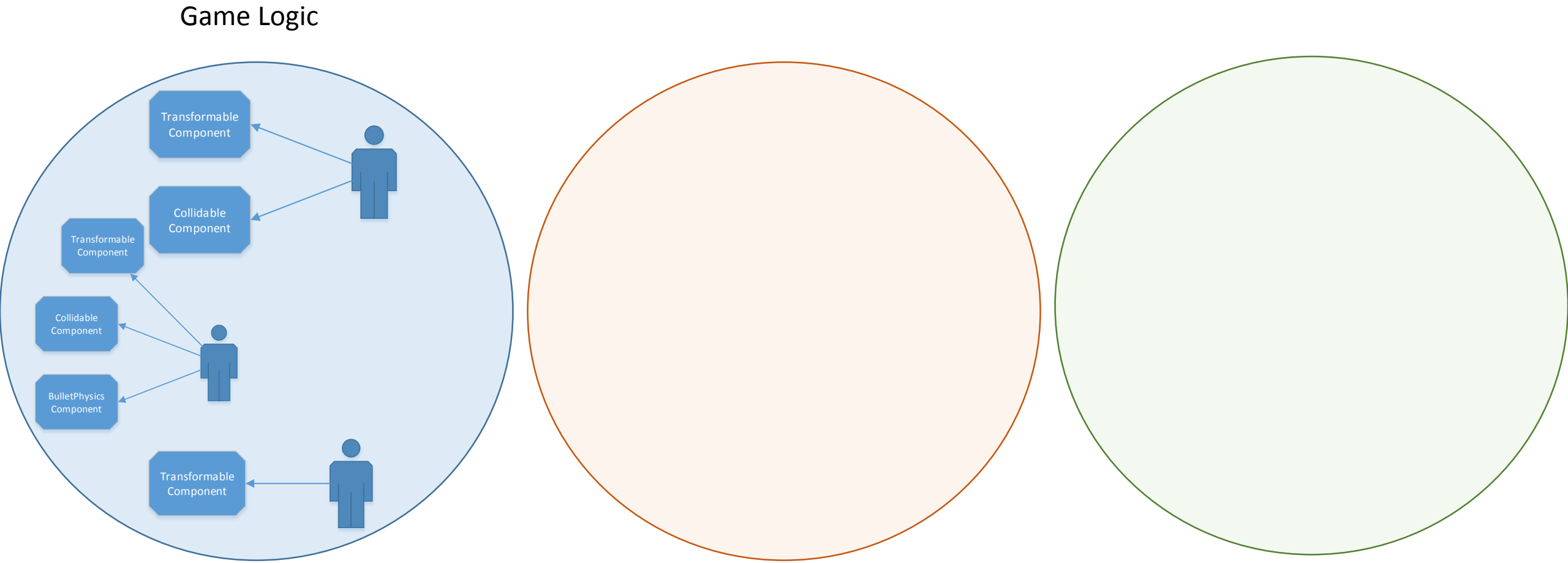
Components

- In UGE, components are small, self-constricted classes that stores game data. When a component is attached to an actor, the actors acquires the data members of the component.
  This way, the actors also acquires the behaviors of its components. For instance:
  - An actor with a `TransformableComponent` attached has a position, a scale and a orientation in the game world. This means the Game Logic is able to move, resize or rotate it;
  - An actor with a `CollidableComponent` attached participates in physics collisions.
- It is possible to attach and detach components from an actor during run-time. Thus, it is also possible to change the behavior of an actor whilst the game is running.
- Currently, saving a few exceptions, all UGE components are data-only.

Components


Game Logic
IO-Free
Definition
Actor Archetype
XML Resource
Transformable
Component
Collidable
Component
Actor:
Airplane
Position, Orientation, Scale
and Collision Shape in the
Game World

A Game World has Many Actors



A Game World has Many Actors

- A game world usually has tens to thousands of game actors.
- In UGE, what distinguishes an actor from another is the `ActorID` and the attached component.
- Using a data-driven architecture, it is possible to define an archetype to create actors of a type – it is like a class declared in an external resource. In UGE, it is a XML file.
- For instance, an actor with a world transform, a collision shape, physics information and health can be defined as…

A Game World has Many Actors

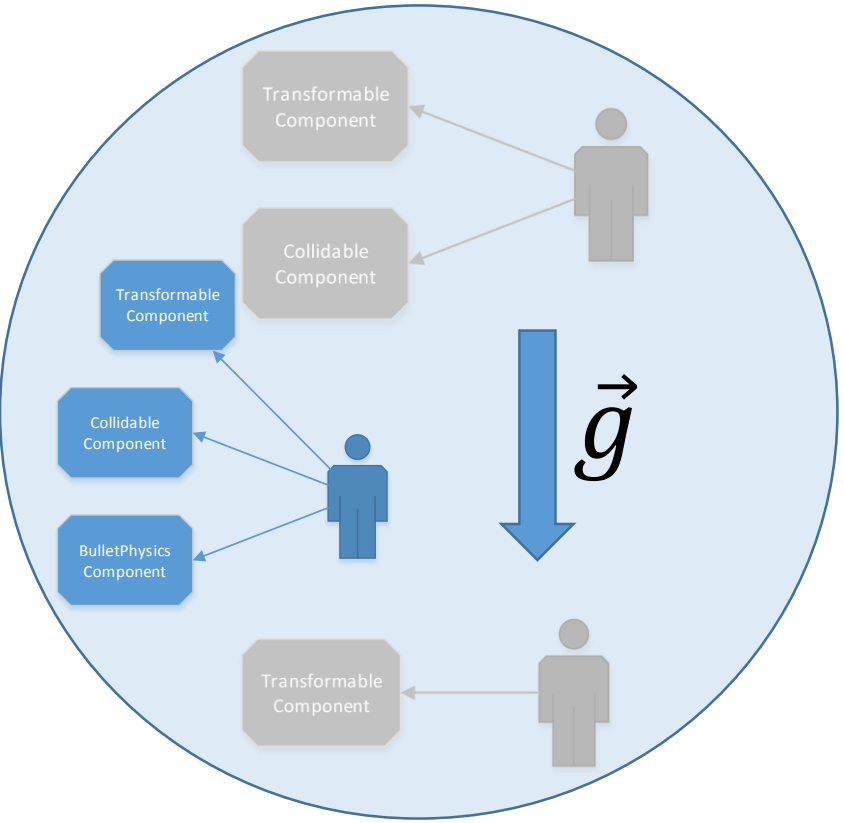
```
<?xml version="1.0" encoding="UTF-8"?>
<Actor type="Airplane" resource="airplane.xml">
  <TransformableComponent>
    <Position x="0.0f" y="0.0f" z="180.0f"/>
    <!-- YXZ order (yaw, pitch, roll), in radians -->
    <Rotation yaw="3.1415f" pitch="0.0f" roll="0.0f"/>
    <Scale x="15.0f" y ="10.0f" z="30.0"/>
  </TransformableComponent>
  <CollidableComponent>
    <Shape type="Box">
      <Dimension x="1.0f" y="1.0f" z="1.0f"/>
    </Shape>
    <CenterOfMassOffset>
      <Position x="0.0f" y="0.0f" z="0.0f"/>
      <Rotation yaw="0.0f" pitch="0.0f" roll="0.0f"/>
    </CenterOfMassOffset>
    <Density type="Pine"/>
    <Material type="Elastic"/>
  </CollidableComponent>
  <BulletPhysicsComponent>
    <LinearFactor x="1.0f" y="1.0f" z="1.0f"/>
    <AngularFactor x="0.0f" y="0.0f" z="0.0f"/>
    <MaxVelocity v="15.0f"/>
    <MaxAngularVelocity v="0.0f"/>
  </BulletPhysicsComponent>
  <HealthComponent>
    <InitialHealthPoints value="100"/>
    <MaximumHealthPoints value="100"/>
  </HealthComponent>
</Actor>
```

A Game World has Many Actors

- To create an actor similar to this one, it is only necessary to use this resource as the archetype.
- To change the actor's initial value, it is only necessary to change the external resource file. Thus, it is not necessary to recompile the game project, reducing iterations time.
- It is possible to override the default values from the stereotype and create a custom actor. For instance, it is possible to attach an output component to an IO-free archetype. This is covered later.

A Game World has Mechanics and Rules

Game Logic

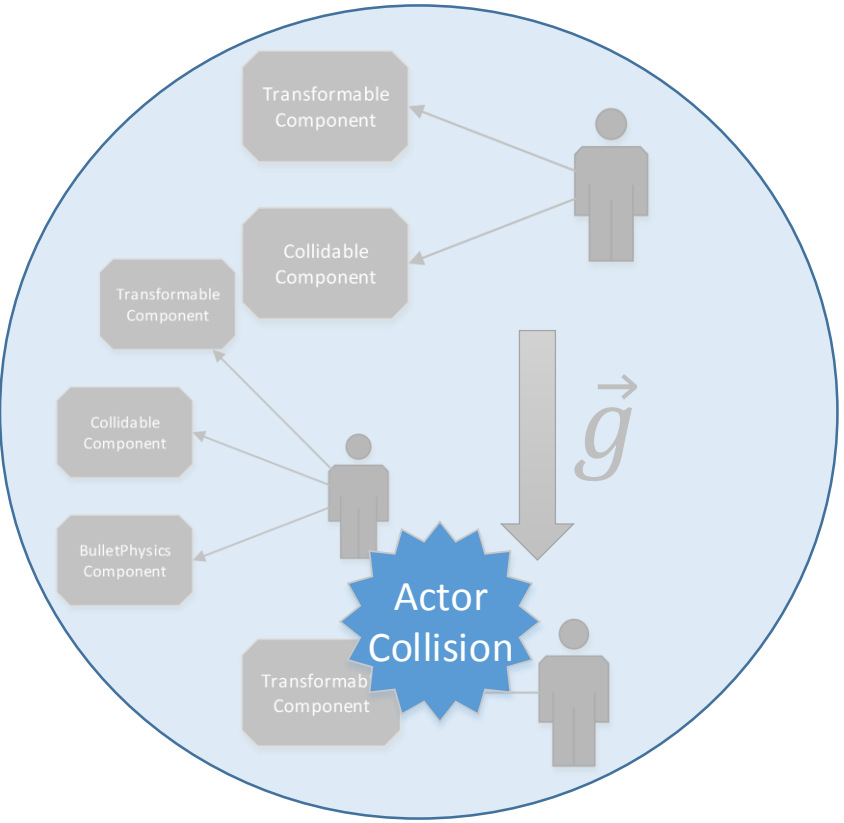


A Game World has Mechanics and Rules

- Every game has its own mechanics and rules. They define the gameplay and what actions the actors can perform.
- For instance, last slide added a gravitational acceleration to the game world. With this acceleration, any actors with a `TransformableComponent`, a `CollidableComponent` and a `BulletPhysicsComponent` will suffer the influences of the acceleration and have their velocities changed.
- This situation and many other situations have side effects to the game.

Relevant Interactions Triggers Events

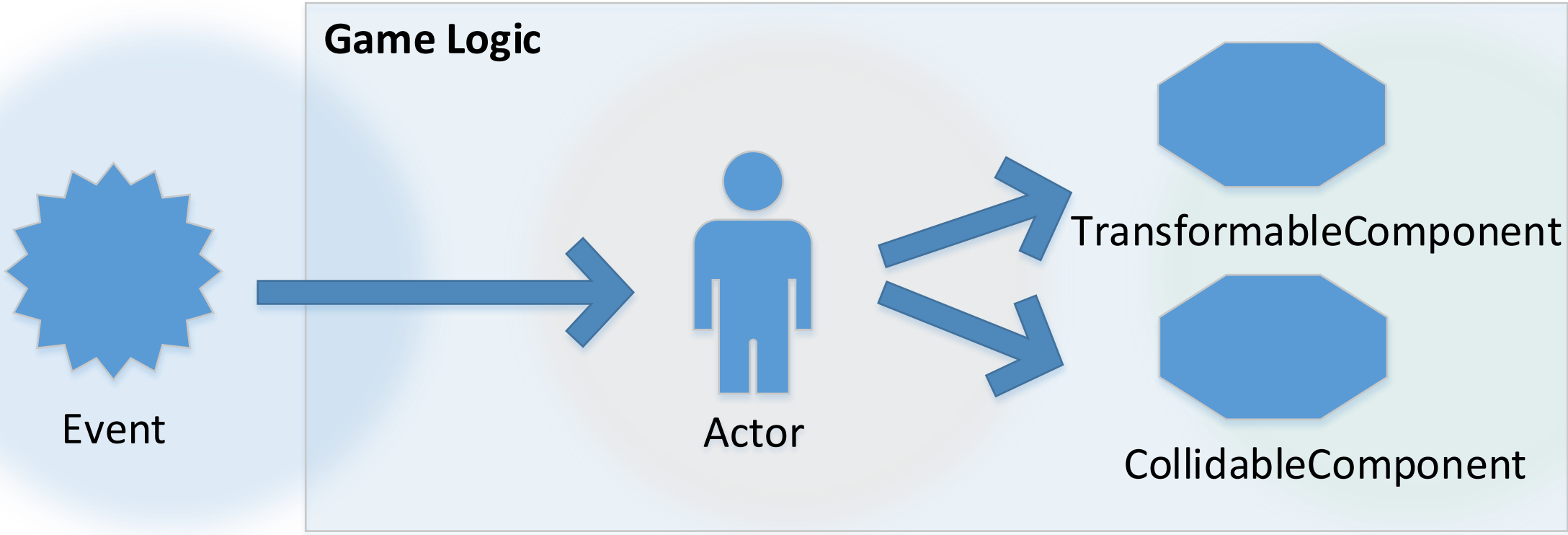
Game Logic



Relevant Interactions Triggers Events

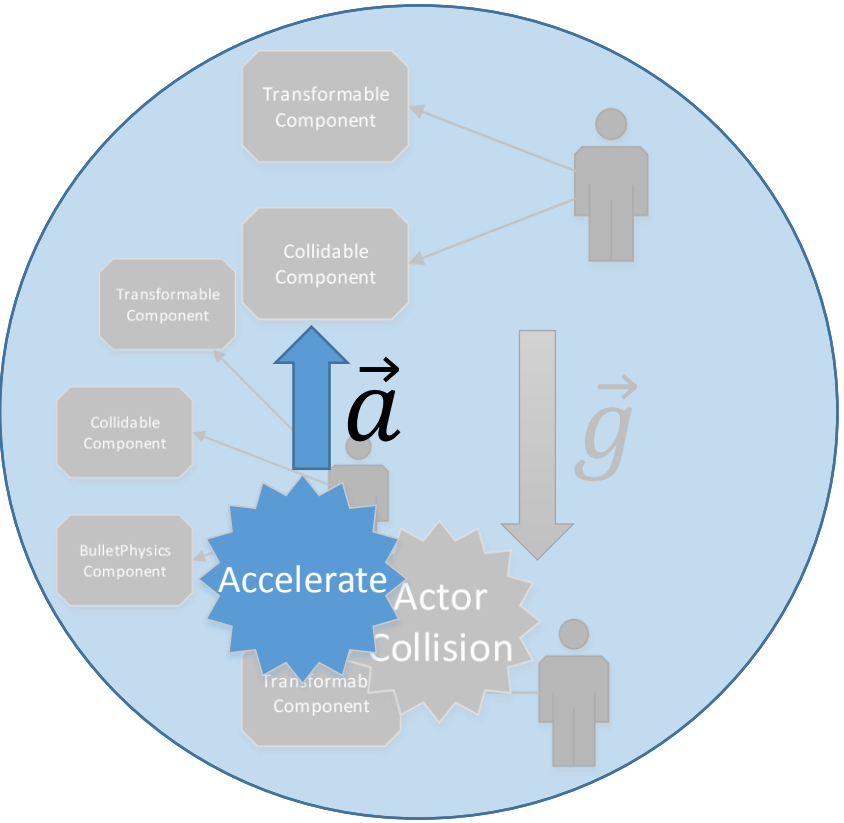
- UGE is an event-driven game engine. As such, any important game action, activity or happenings should trigger an event.
- Events should be defined for relevant Game Logic interactions or side effects.
  - Some examples are collisions between actors, new actors, removed actors and bullet hits.
- In a game accessibility context, events are very important, as they allow different parts of the implementation to handle an important situation according to their goals.
- They also makes the Game Logic implementation more flexible and decoupled: it does not matter where an event was dispatched; if the Game Logic receives an event it is listening to, it will handle the event.

Relevant Interactions Triggers Events



Some Events are Game Commands

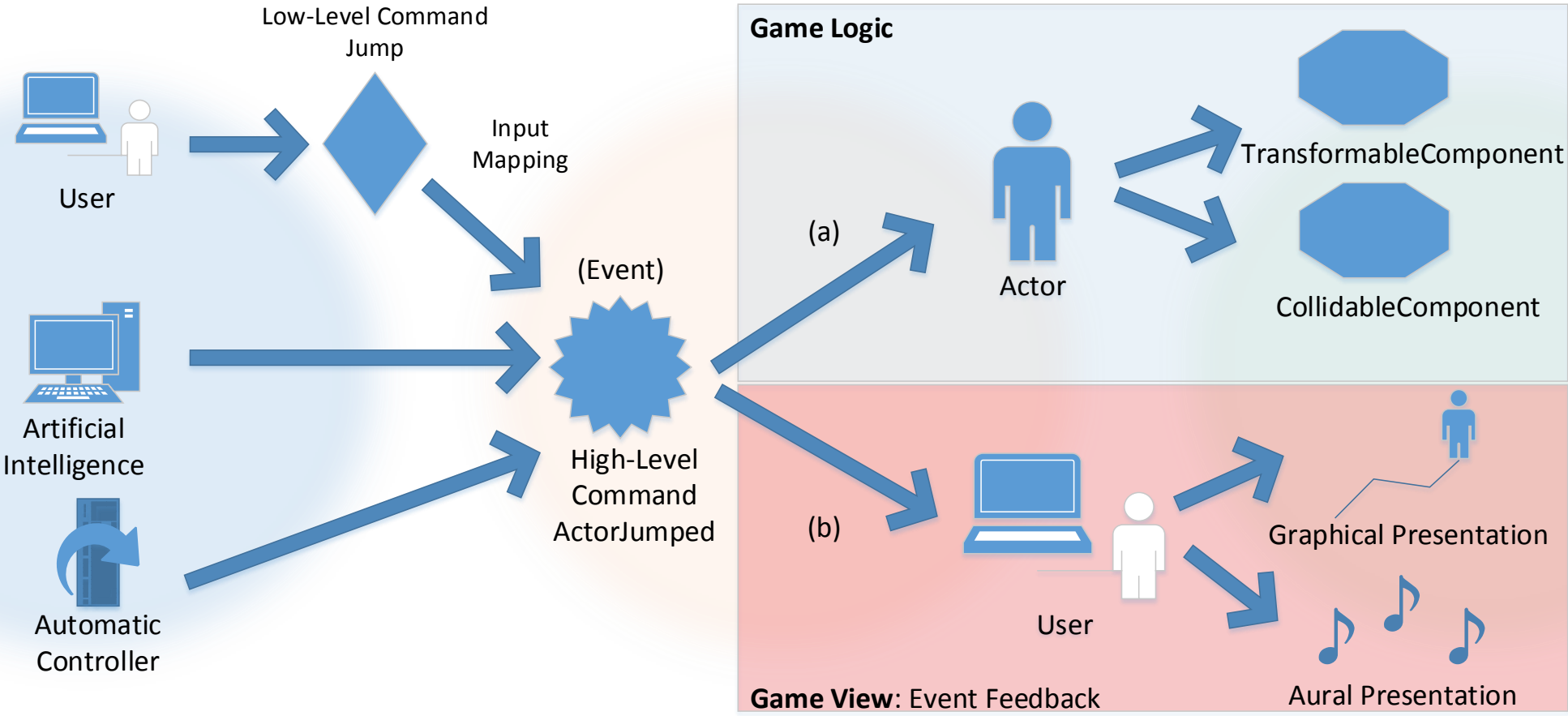
Game Logic



Some Events are Game Commands

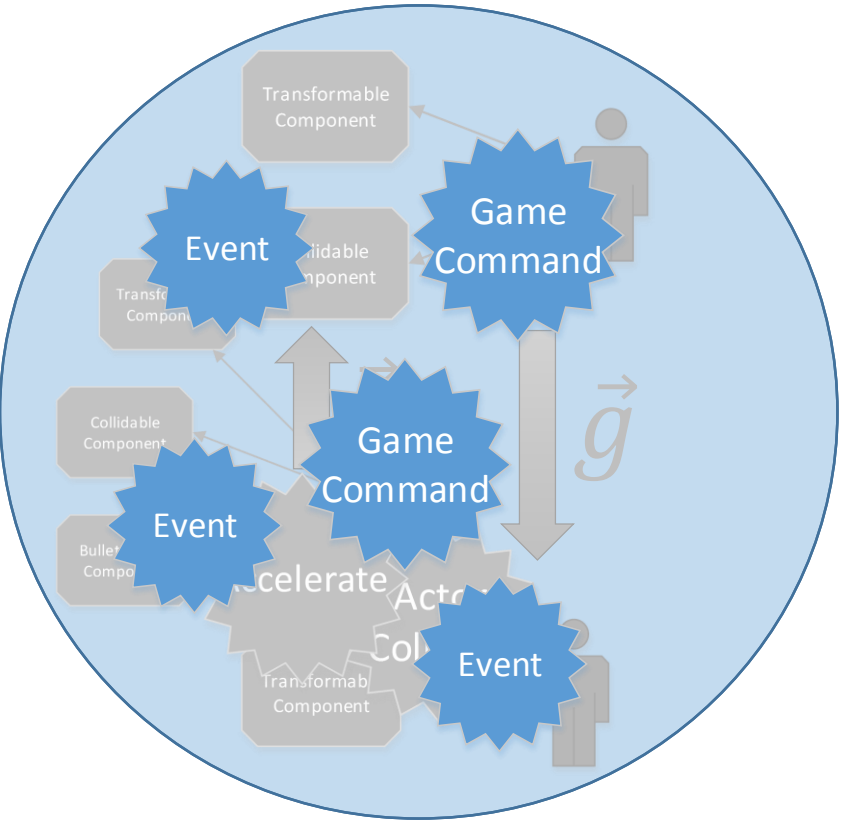
- As the Game Logic is event-driven, it is possible to control the game actors using events. UGE calls this kind of event a game command.
  - For instance, in the previous slide, an Accelerate event requested the Game Logic to increase the acceleration of the actor.
- As a game command is an event, they have all the benefits of events.
- However, there is a less evident benefit: a game command can be issued anywhere, by anyone. This means that:
  - The player controller can send game commands;
  - Artificial Intelligence (AI) actors can send game commands;
  - It is possible to automate the generation of game commands – a bot within the game.

Some Events are Game Commands



Events Rule the Game Simulation

Game Logic



Events Rule the Game Simulation

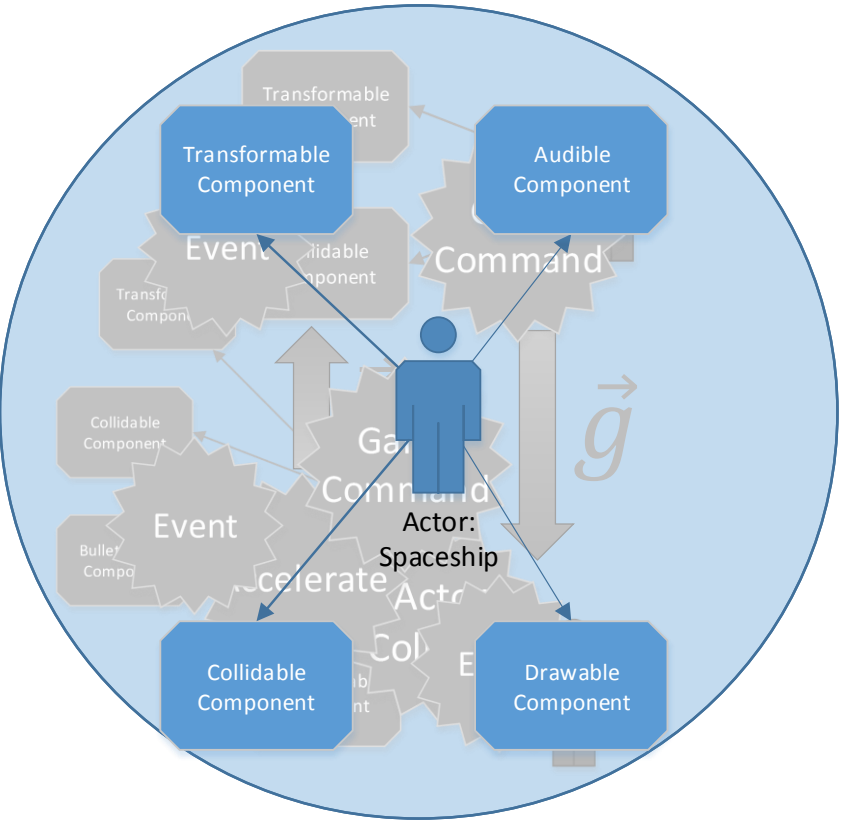
1. As it is possible to control all game actors with events, it is possible to control the game without an user.
   - On one hand, this makes the Game Logic input free.
   - On the other hand, it turns the game into in a simulation*!
2. Considering the Game Logic does not need any output component to simulate the game world, this also means the Game Logic so far is output free.
   - Sprites, meshes and any other do not contribute directly to the Game Logic. Their physic rigid/soft body counterpart do.

- (1) and (2) imply that the Game Logic is input-output free.

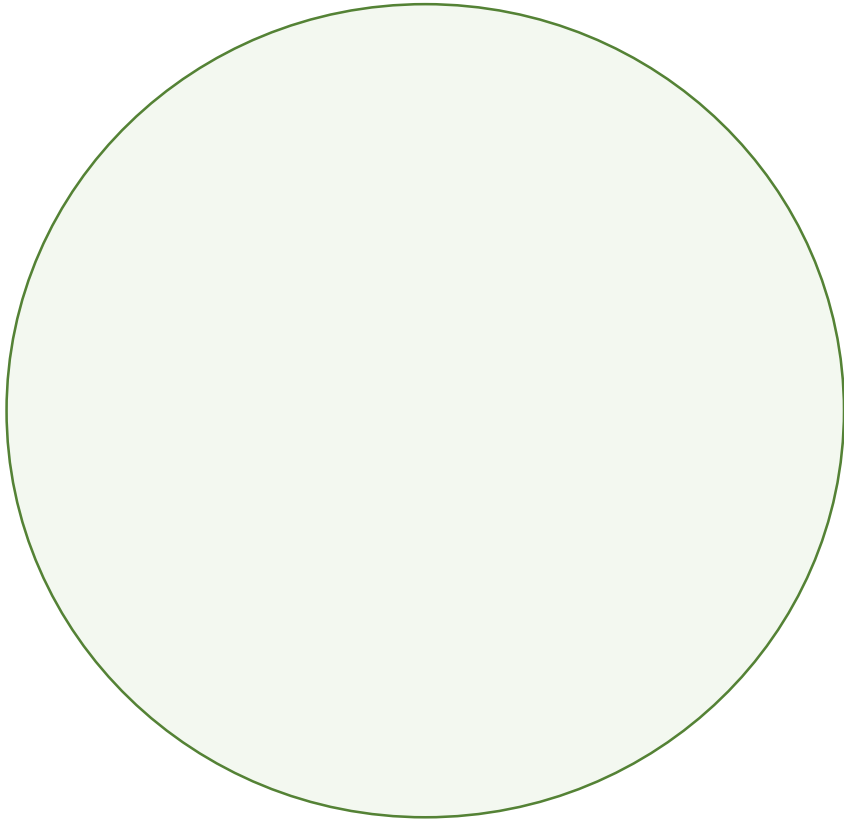
It can also result in an interesting metagame, on which the objective is to provide programmable commands to the game.

Some Components Look Better than Others

Game Logic



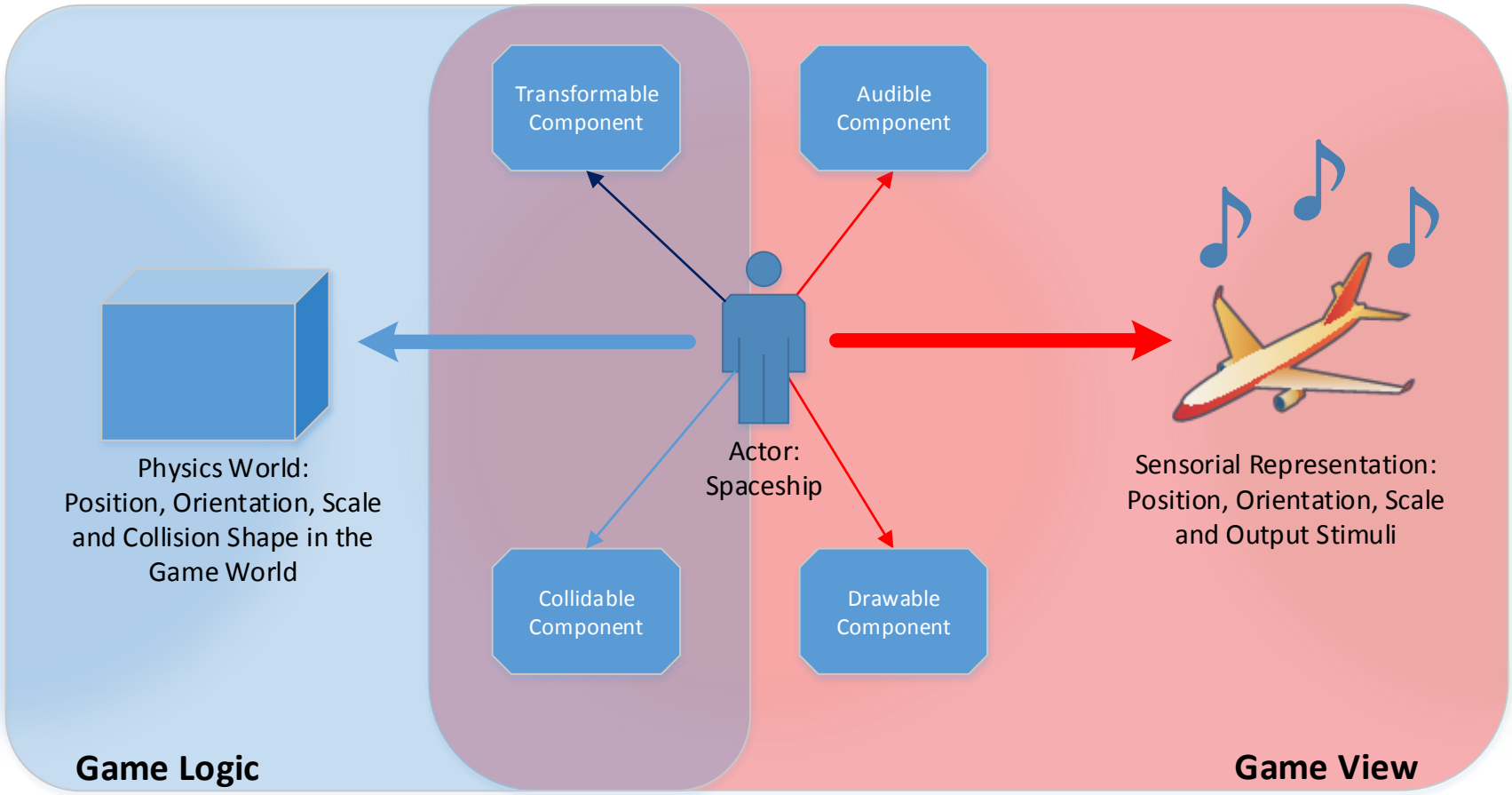
Game View



Some Components Look Better than Others

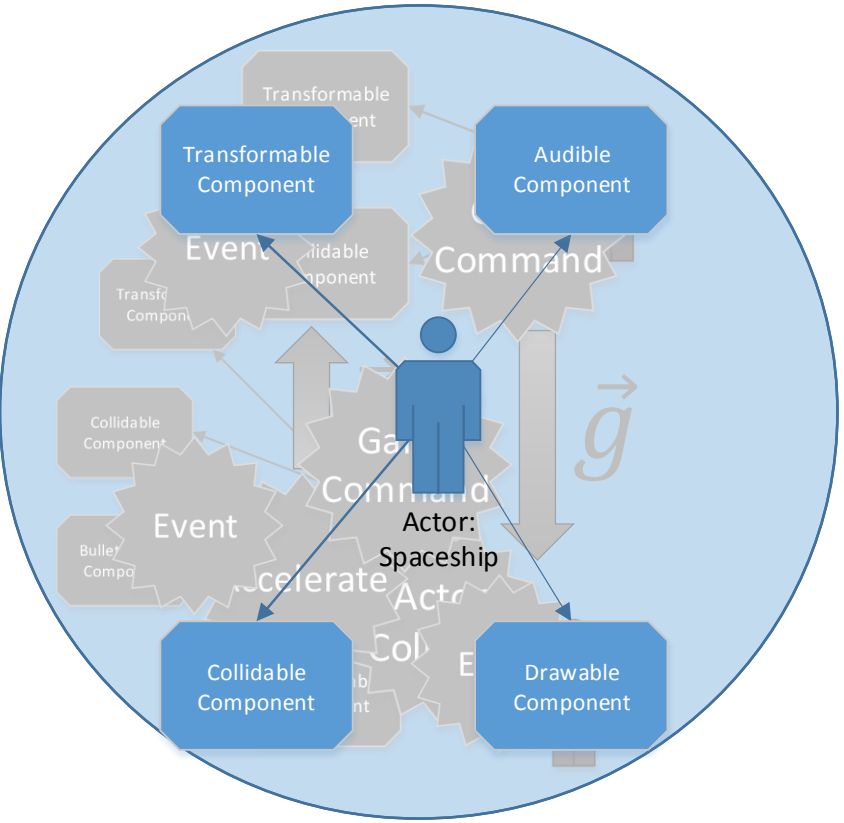
- There finally is an actor with output components. This gives some use to the so far stagnant Game View.
- The Game Logic has an IO-free Scene graph. It handles the `TransformableComponents` of the actors to position them into the game world.
- When an output component is attached to an actor, UGE register this actor to the `SceneRenderer`, which is responsible to translate the components data into output data and provide it to output subsystems. The output subsystems use this data to present sensory stimuli to the user.
- To keep the Game Logic IO-free and ease the creation of new Game Views, the output components are attached in a game specialization steps regarding the abilities of the user.

Some Components Look Better than Others



The Game is Played by a User

Game Logic



Game View

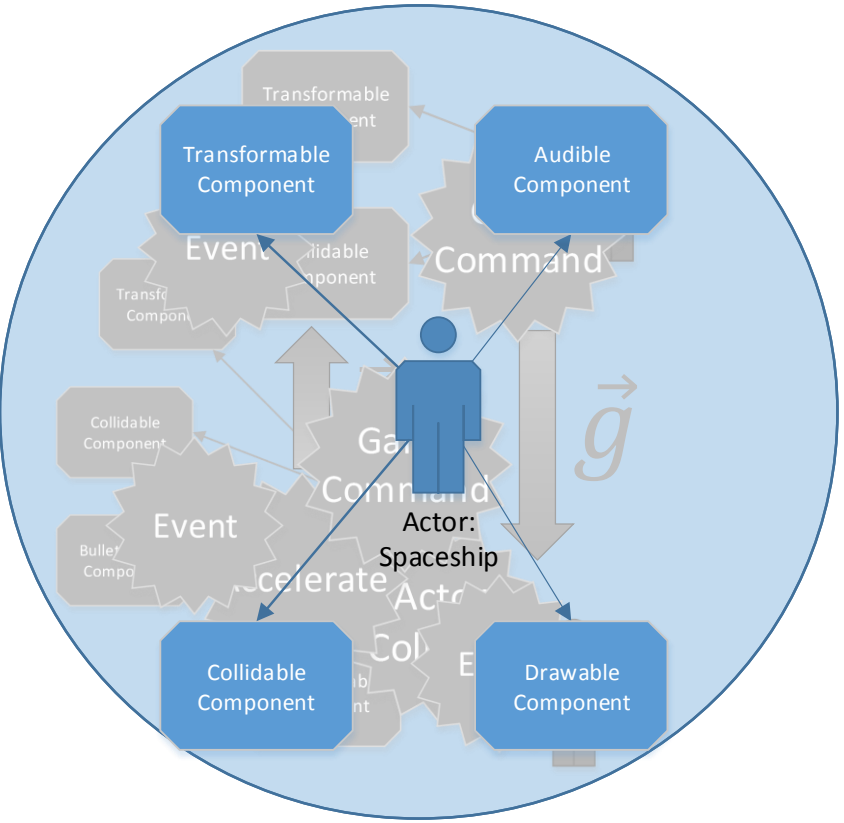
User

The Game is Played by a User

- The main element of a game is not the game itself; it is the user.
- Games requires a set of required interaction abilities to be played. Most games targets the average user abilities.
  - This assumes visual, hearing, motor and cognitive abilities.
  - Often this also assumes the users know a specific language.
- Are such restrictive assumptions adequate to every user?

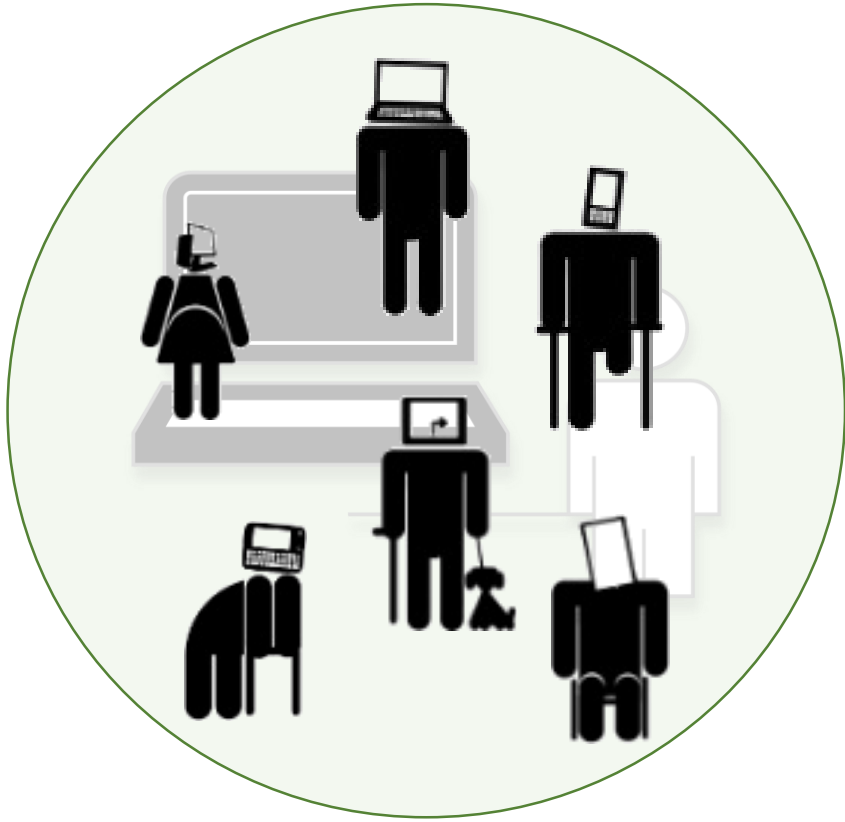
Everyone is Unique

Game Logic



Game View

User

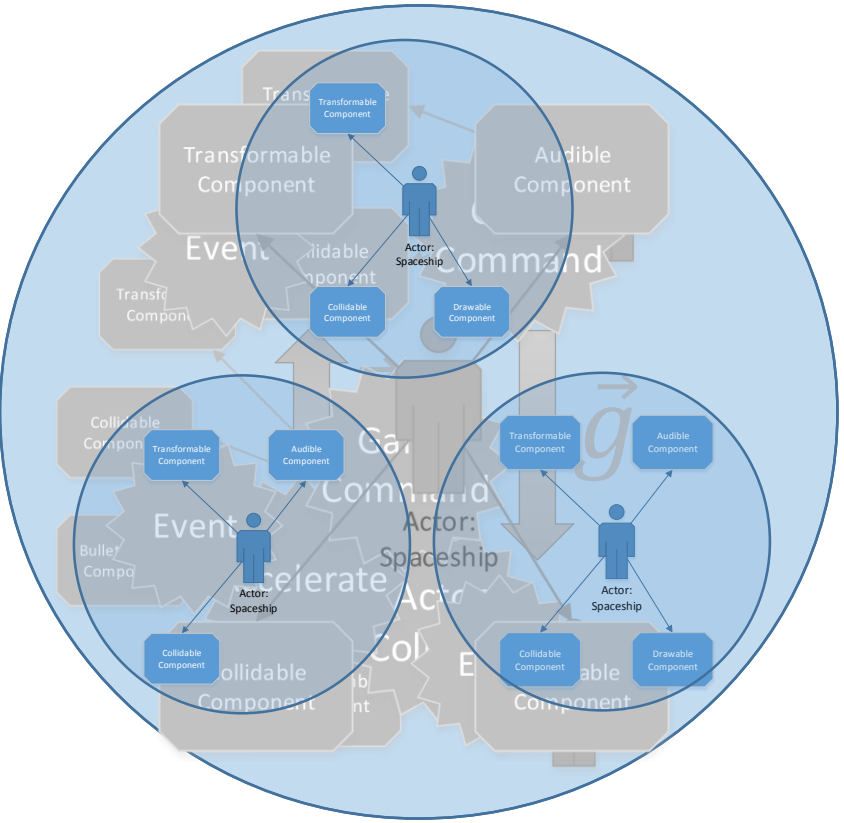


Everyone is Unique

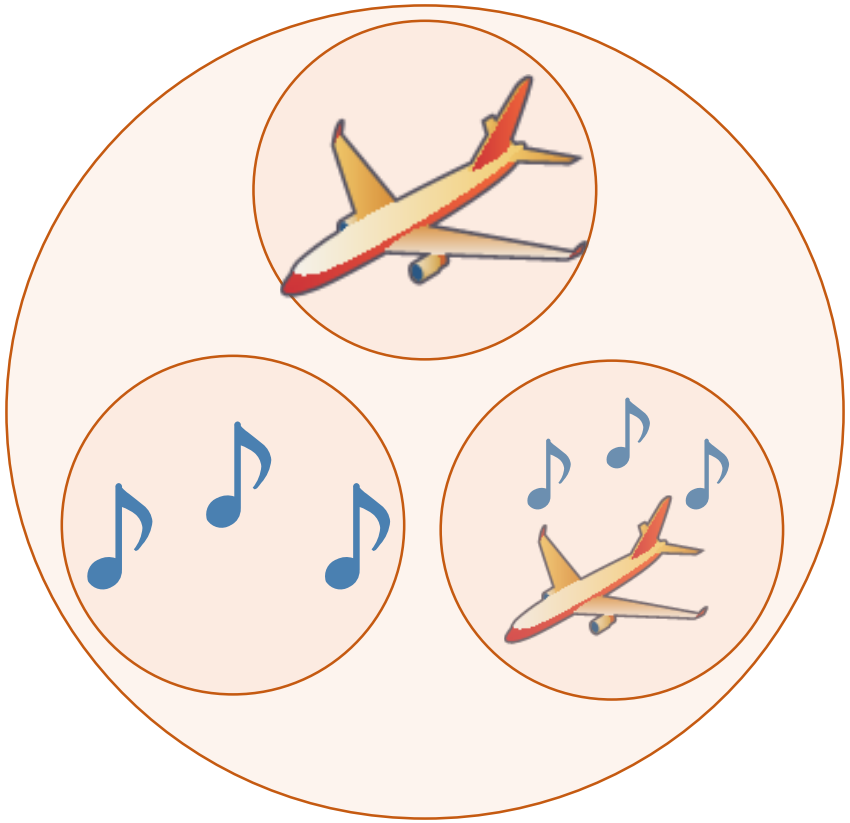
- The interaction abilities of users may greatly vary.
  - For instance, consider how a visually-impaired user interacts with a game;
  - Next, consider how a motor-impaired user would interact with the same game;
  - Finally, consider how you interact with the game.
- All these scenario are very different.
- Implementing a game accessible to everyone is impossible. However, the literature has shown it is possible to implement simple games accessible for a wider public (for instance, Access Invaders).
- So far, this guide covered output specialization in an IO free Game Logic. What if the game could further tailor itself to suit the interaction abilities of a group of users?

Component Specialization

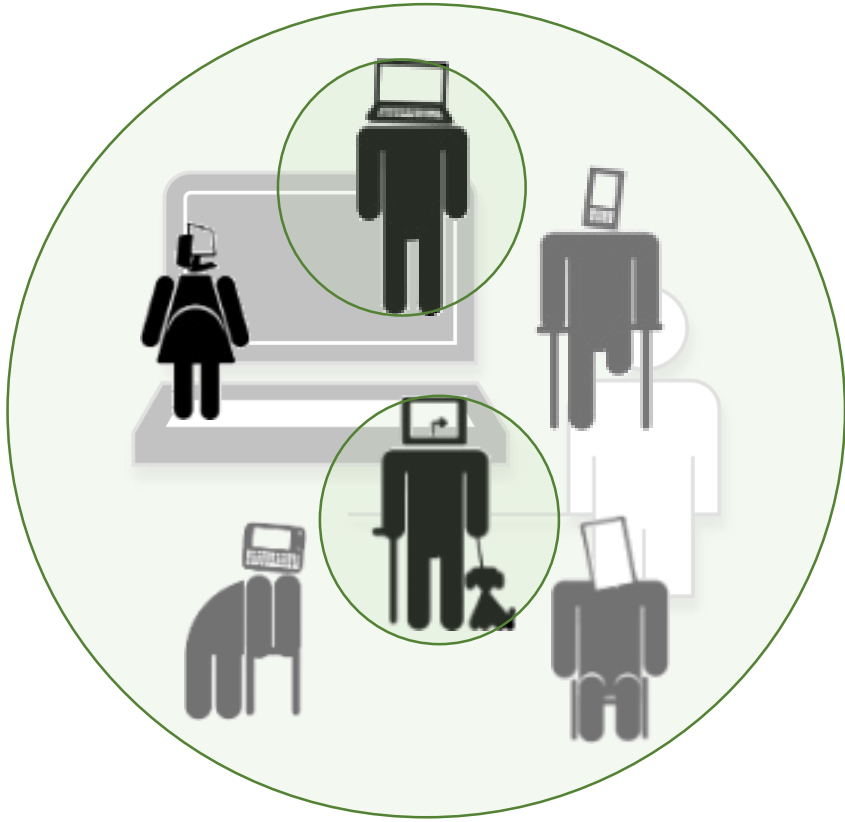
Game Logic



Game View



User



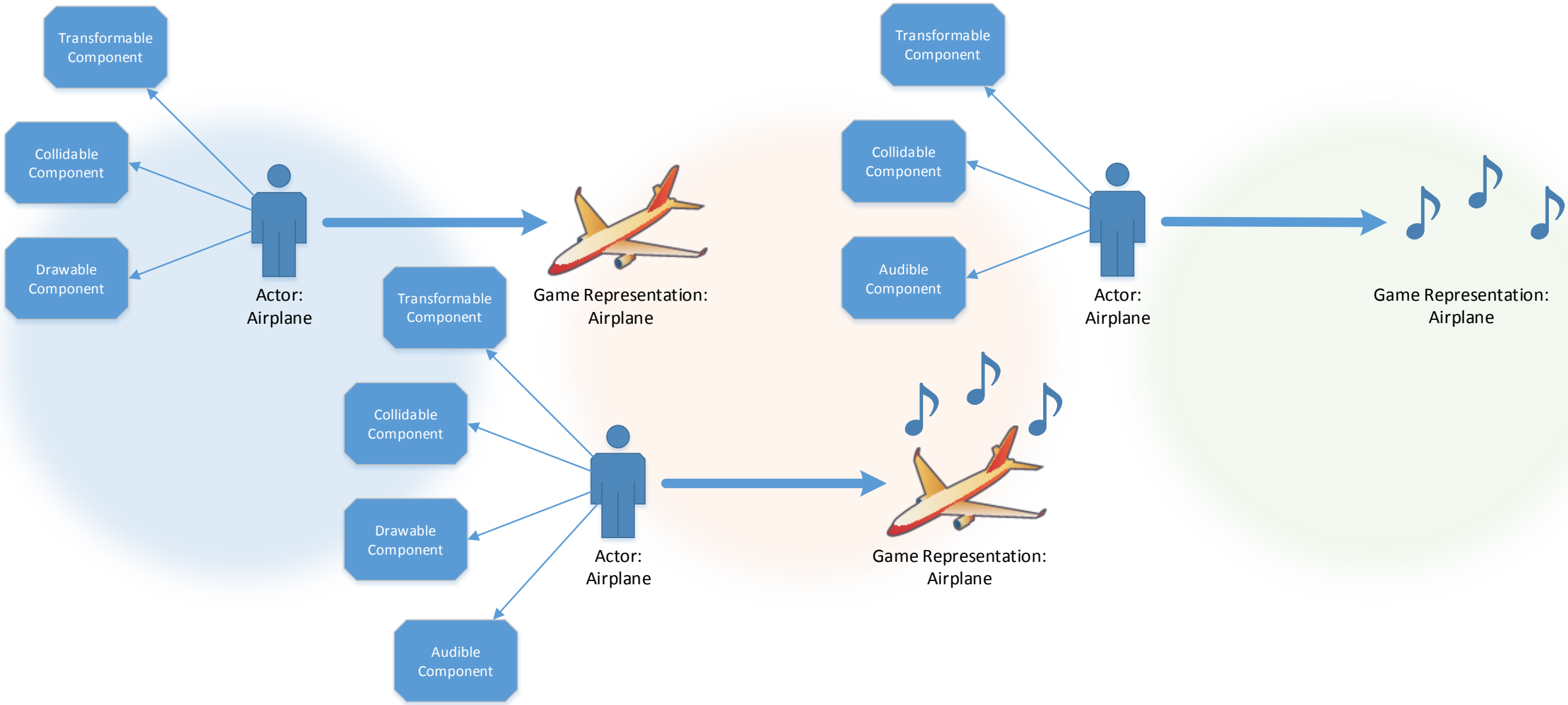
Component Specialization

- As described previously, it is possible to attach output components to add stimuli output.
  - It is equally possible to remove (or just not to attach) the component to remove a stimuli output.
- Thus, component specialization may aid implementing accessible versions of the game for some sensorial disabilities. For instance:
  - A `DrawableComponent` adds visual stimuli, whilst an `AudibleComponent` adds aural stimuli.
  - Increasing the scale of the `TransformableComponent` aids creating extra-large models for low-vision users.

Component Specialization

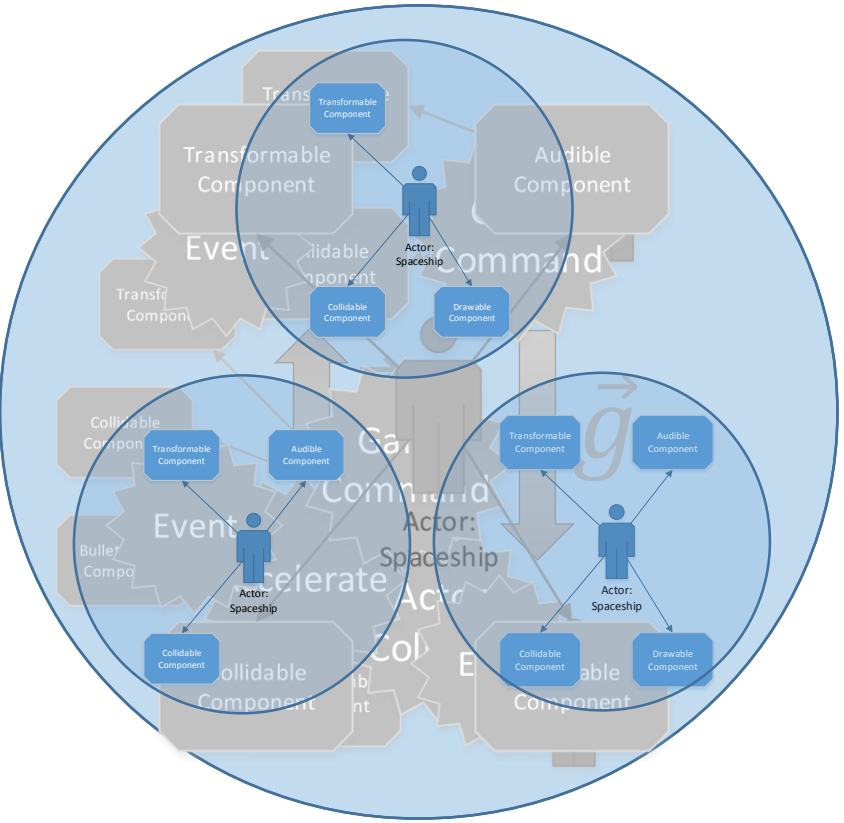
- Tweaking the components data may also help for some disabilities. This kind of change is usually associated to gameplay changes and contributes to increase or decrease the game difficult. For instance:
  - Power of projectiles;
  - Speed of actors;
  - AI controlled helpers.
- It is important to note that removing an output component or tweaking the data will usually not be enough to automatically create an accessible version of the game.
  - However, it offers a starting point – and not having to re-implement the Game Logic is usually an advantage.

Component Specialization

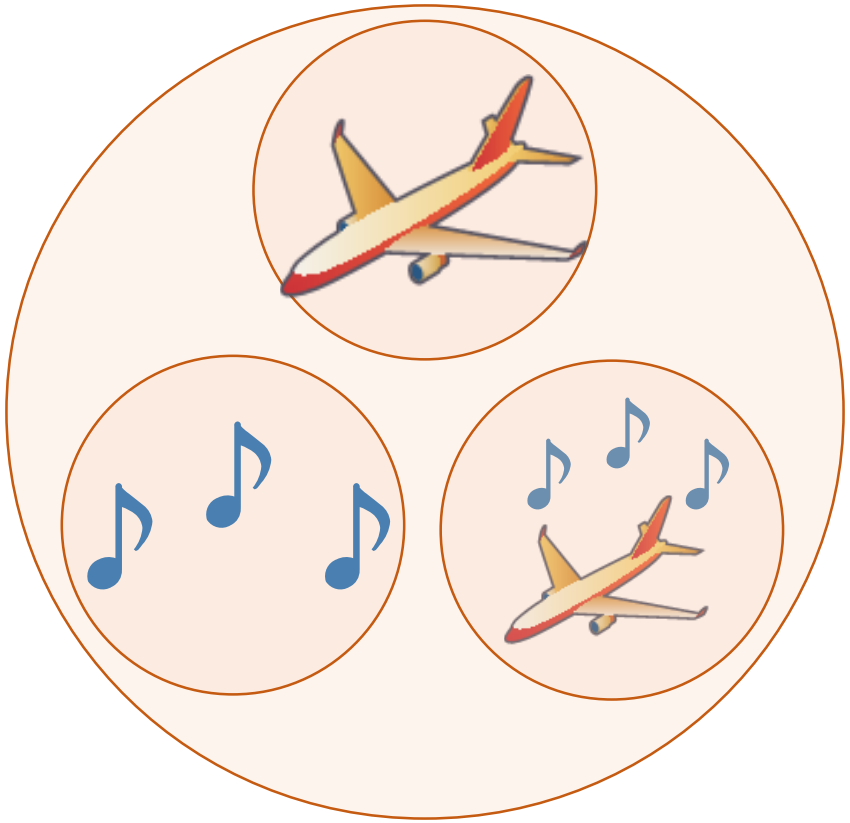


Event Specialization

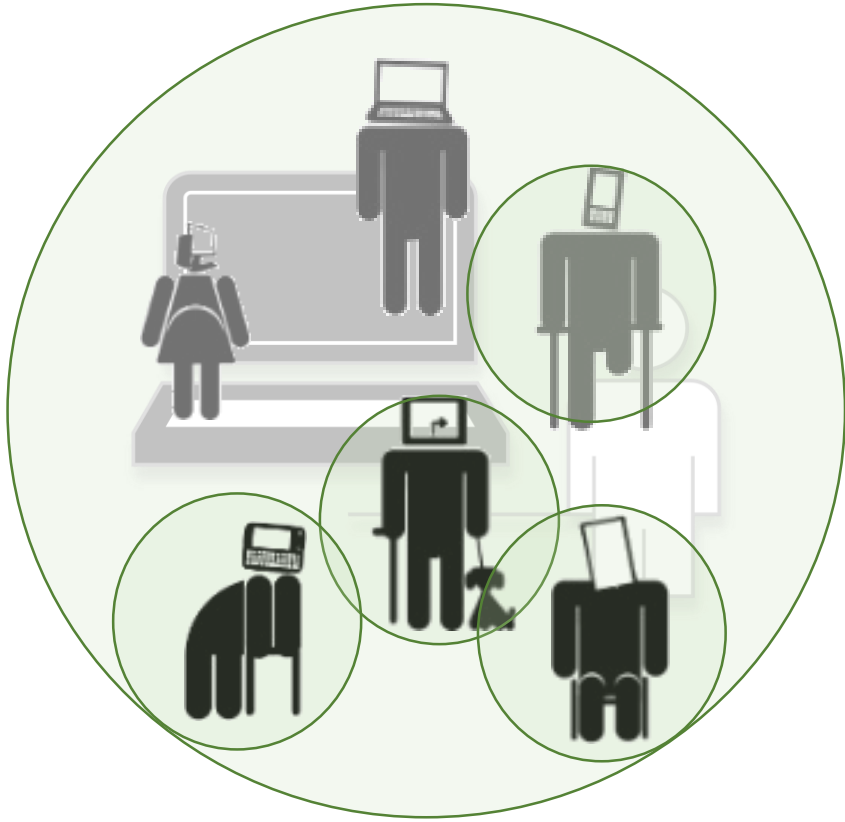
Game Logic



Game View



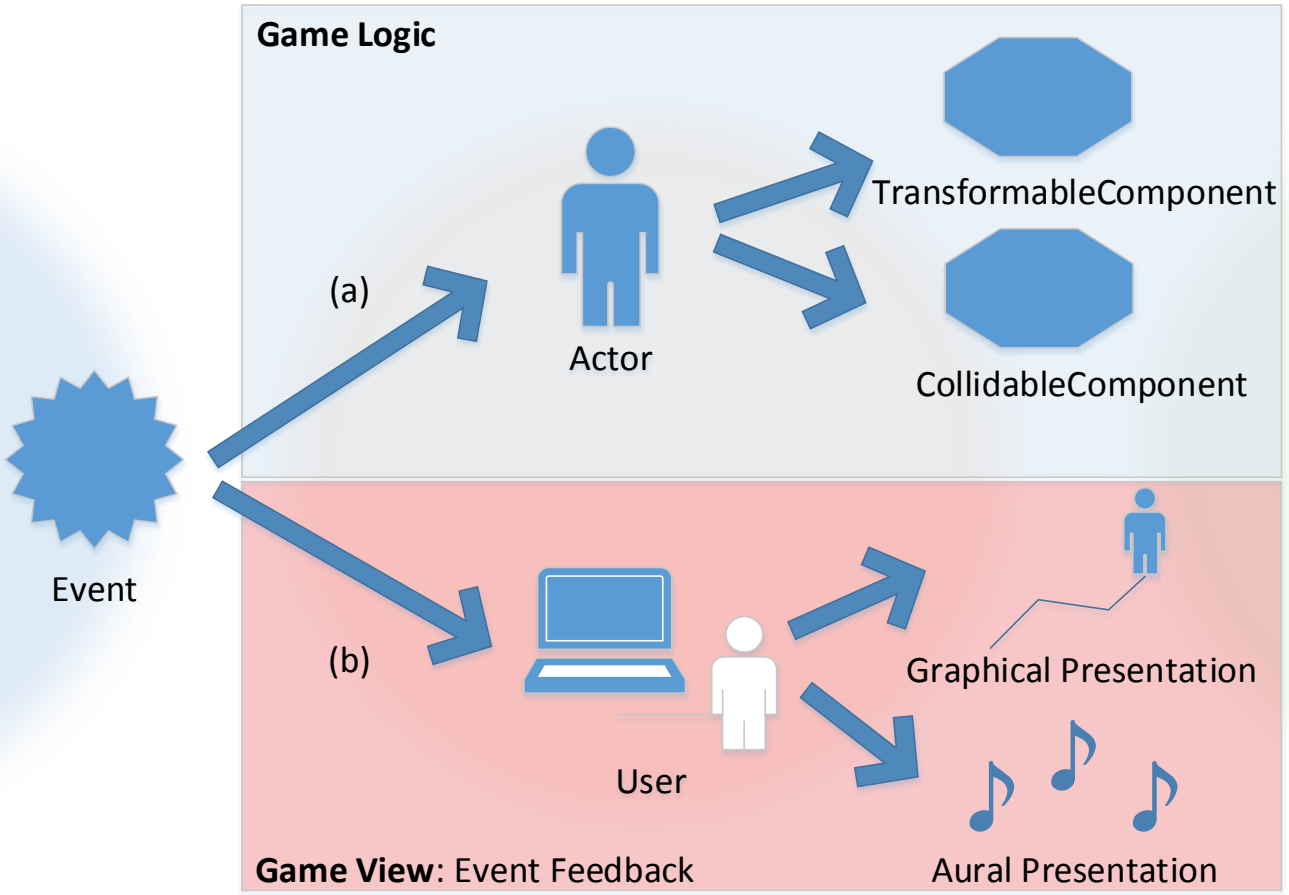
User



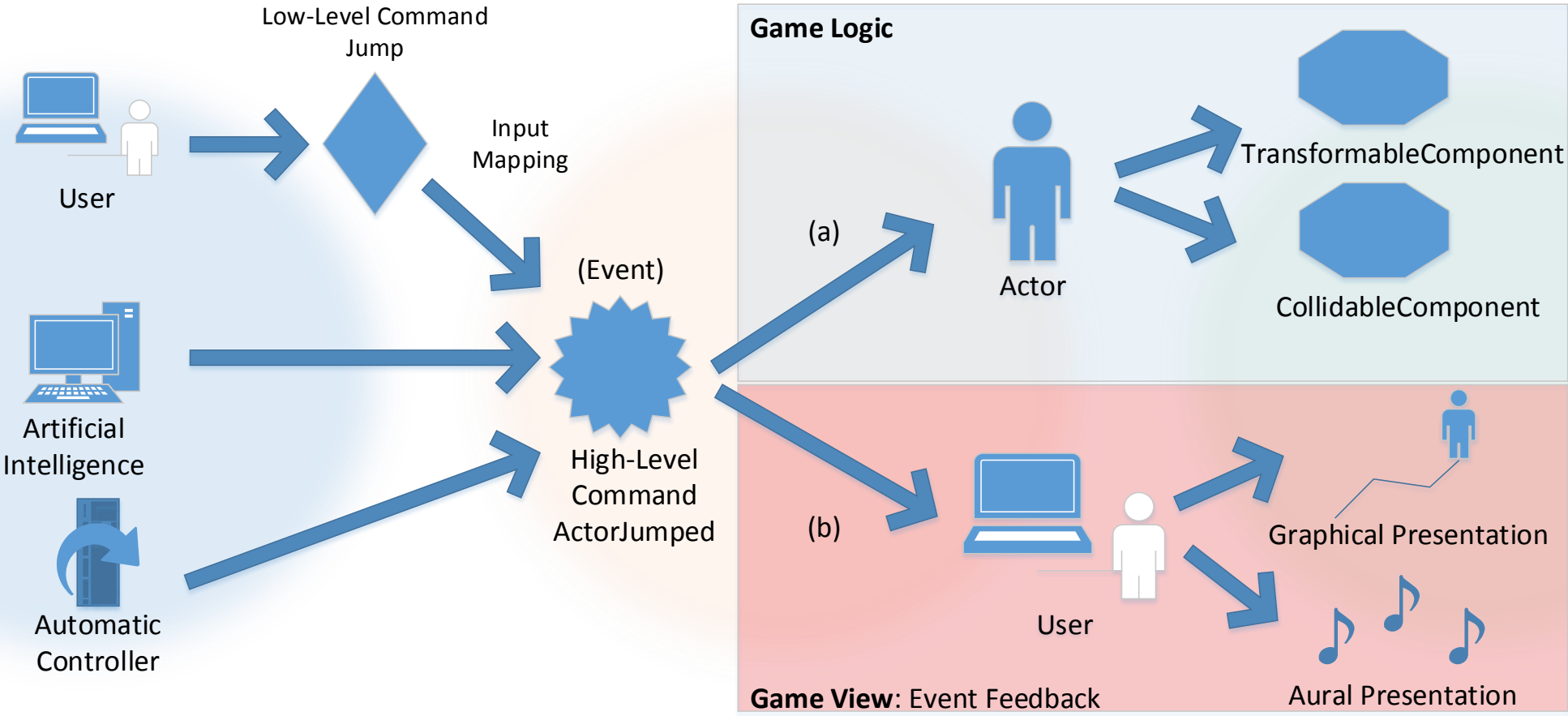
Event Specialization

- The Game Logic is event-driven. This mean the game implementation has already several existing events that can be handled to provide feedback.
  - This contributes to creating an audio-only version of the game and to enhance the extra-large version of the game;
  - It also contributes to creating a deaf version of the game, by exploring graphical effects such as particles or onomatopoeias to provide graphical feedback.
- As mentioned before, game commands provide a unified way to control game actors. With game commands, it is possible to automate certain Game Logic tasks, aiding some users to play.
- This means an UGE Game Controller can either use actions provided from the user or to issue game commands to help the user to play.
  - For instance, allowing an AI to perform certain game commands for the user might be helpful for motor-impaired users;
  - Automation might also be helpful for elderly users on tasks which requires precision.

Event Specialization



Event Specialization

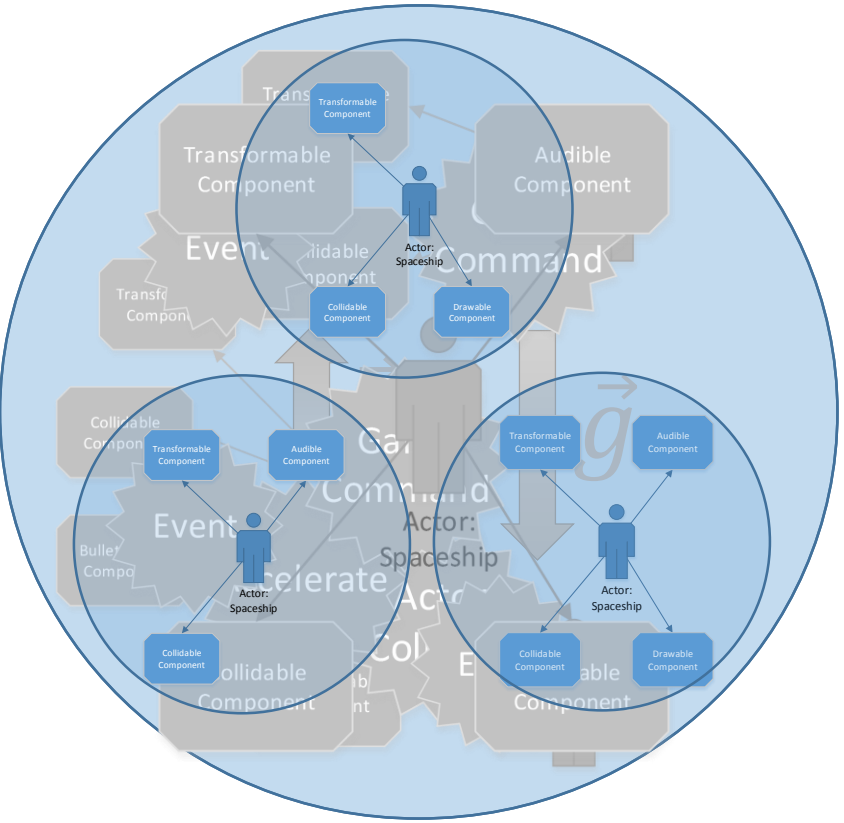


Event Specialization

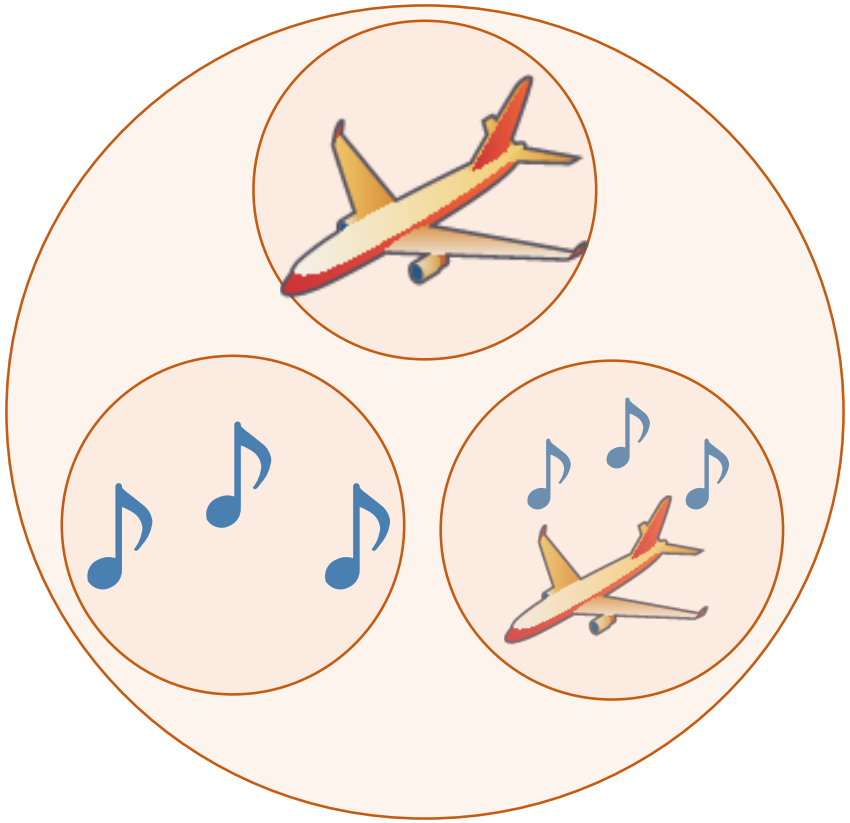
- Event handlers, like components, can be registered during run-time.
  - This allows reusing event handlers in different versions of the game;
  - It also allow the user to choose the event handlers he/she wishes to personalize the game presentation.
- If necessary, it is always possible to create and dispatch new event in the Game Logic with few changes to the codebase.
- Accessibility and usability evaluation might suggest that a stimuli is missing. This may lead to the definition of a new game event. Every time a new event is introduced and handled to provide feedback, it may benefit several game versions, potentially improving the playing experience of many users.

Game View Specialization

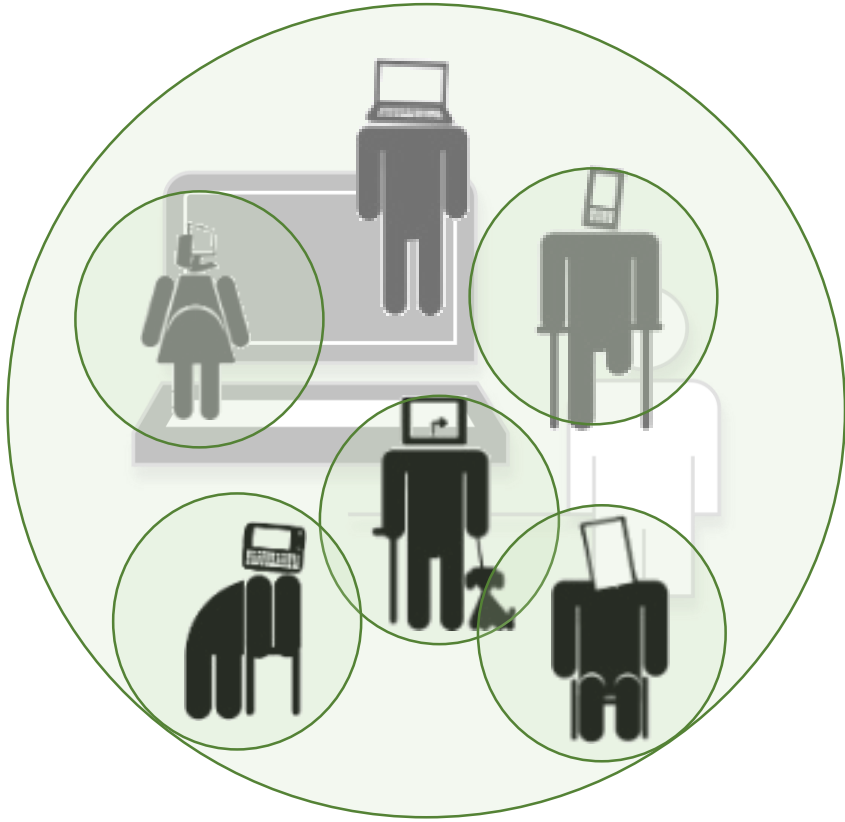
Game Logic



Game View



User

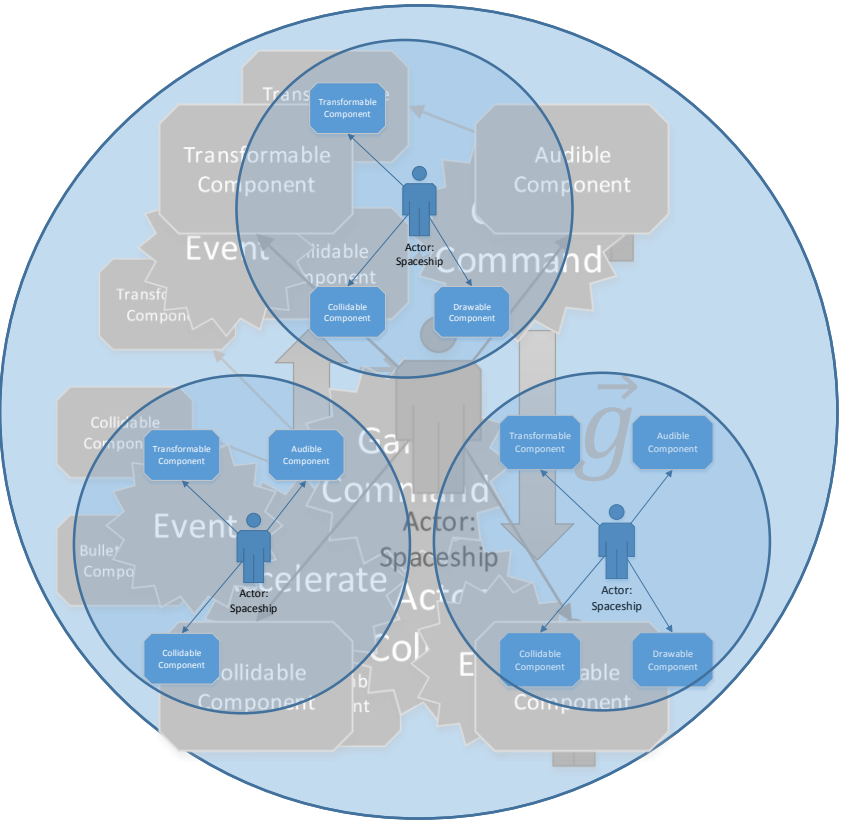


Game View Specialization

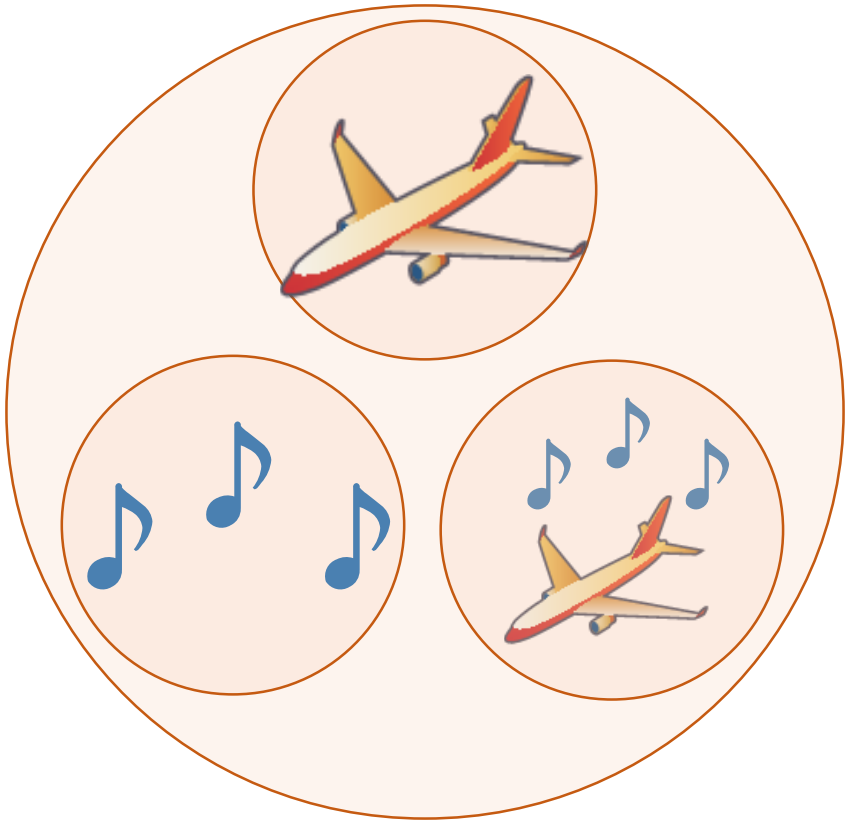
- As the Game Logic does not depends on the Game View, it is always possible to create a new Game View without changing the logic.
- This allows exploring different settings for cameras, view perspectives and game controllers.
  - Those different settings can be useful for visually impaired users.

Player Profile Specialization

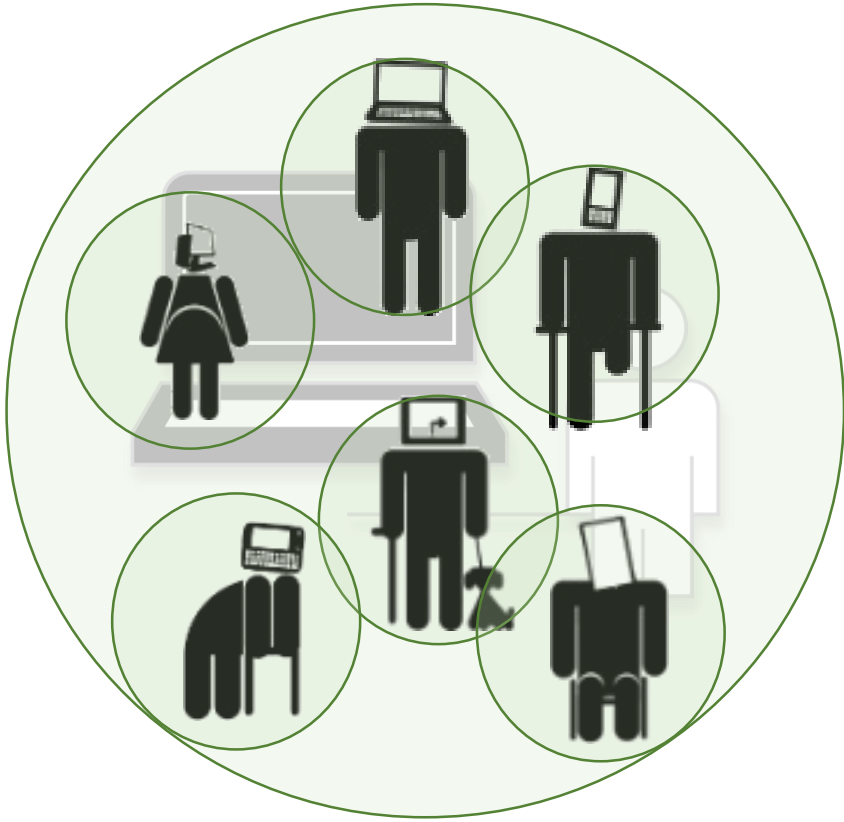
Game Logic



Game View



User



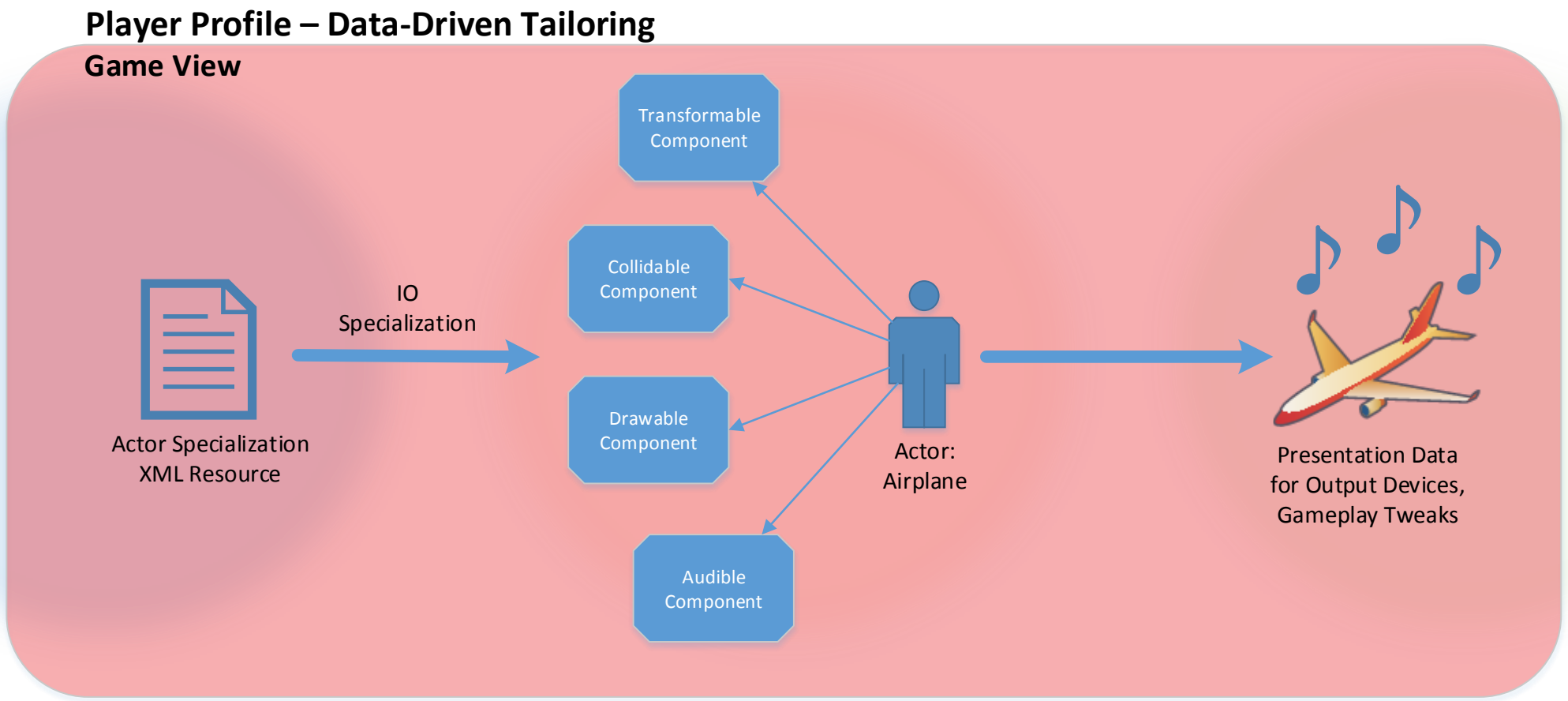
Player Profile Specialization

- As components and events handlers can be chosen and configured during run-time, it is possible to configure the game interaction via pre-defined, external resources.
  - Provided the component is implemented, component specializations can always be data-driven;
  - Event handlers can be enabled or disabled according to the preferences of the user.
- This is the concept of UGE's player profile: an external XML resource file to tailor the game.
- It is possible to create a player profile for a specific user or for a group of user. This allows creating pre-sets for different interaction abilities and customizing it according to the preferences of the user.

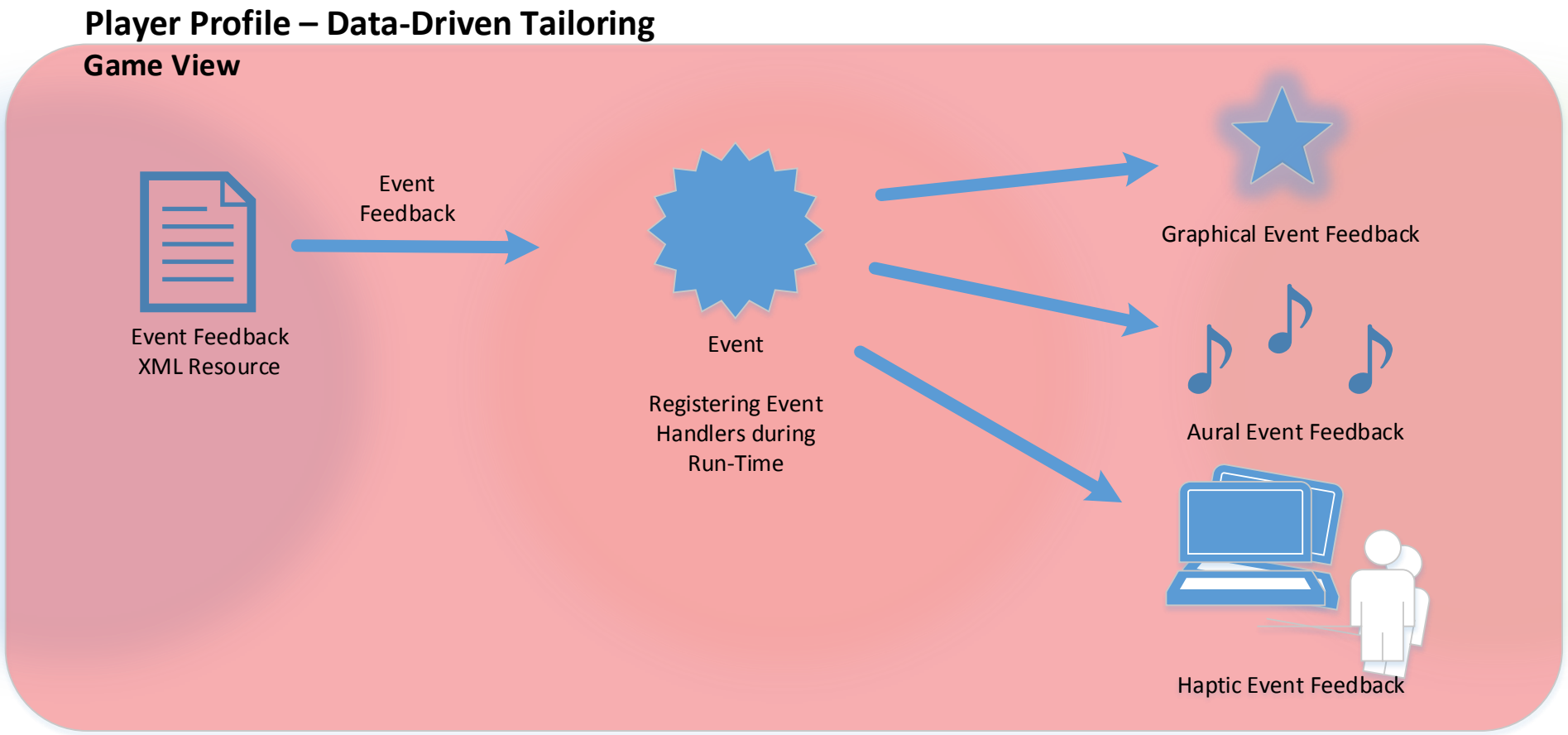
Player Profile Specialization

- The player profiles supports and eases:
  - Defining the desired IO devices for game interaction;
  - IO specialization for the game, both for actors and events;
  - Defining a profile-specific input mapping;
  - Tweaking the gameplay;
  - Reducing the number of different versions of the game.
- UGE supports using a list of player profiles to provide the user different interaction options.

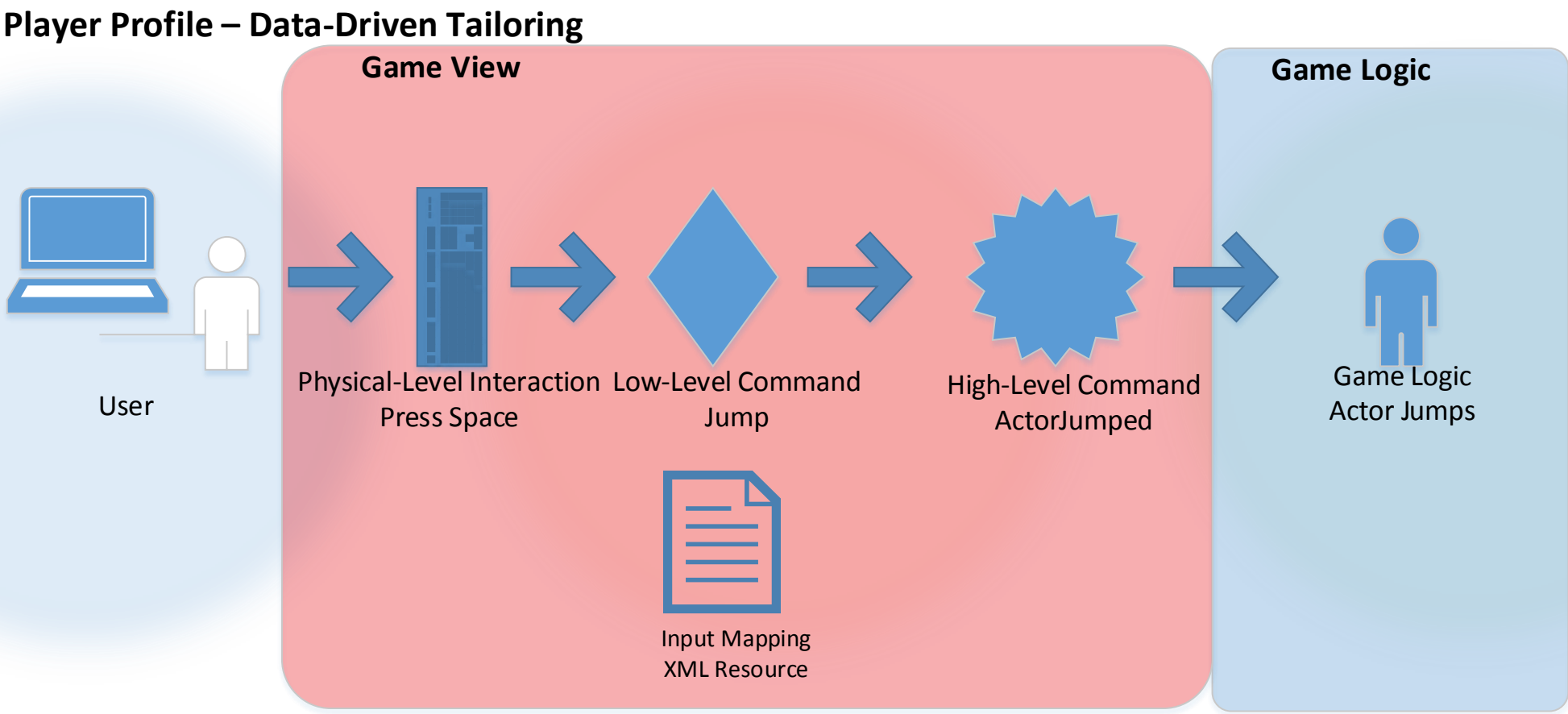
Player Profile Specialization



Player Profile Specialization

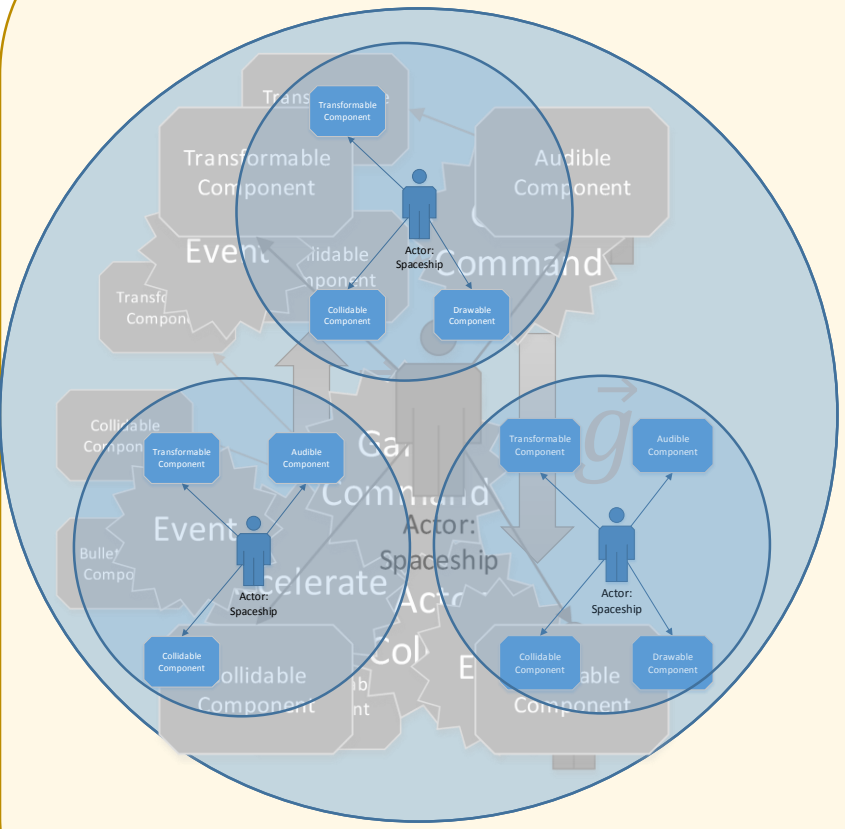


Player Profile Specialization

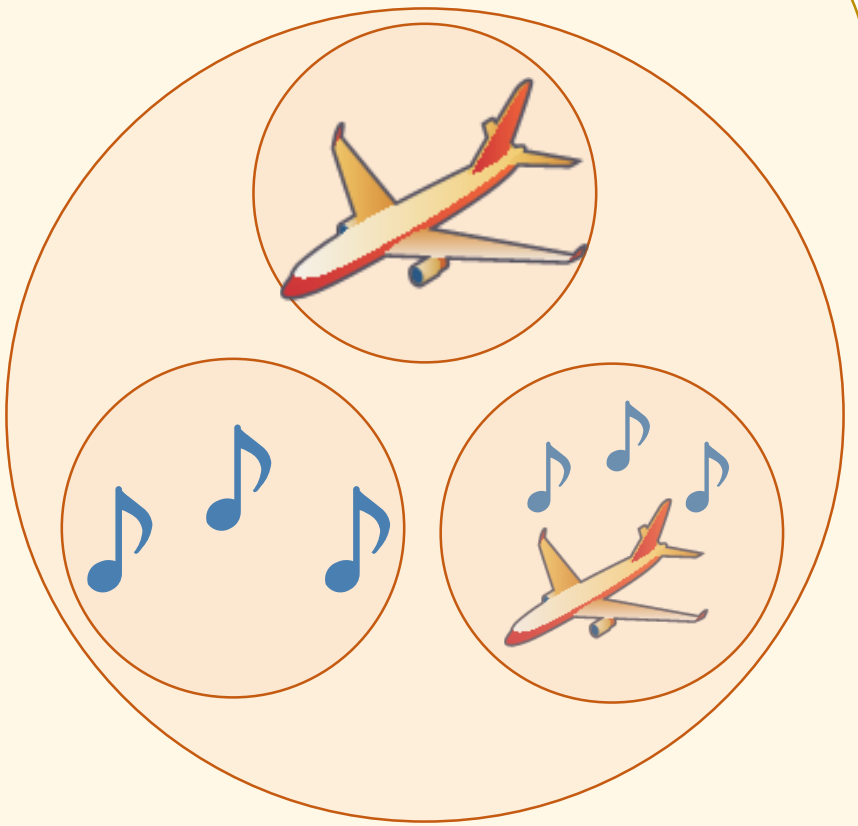


The Game Application

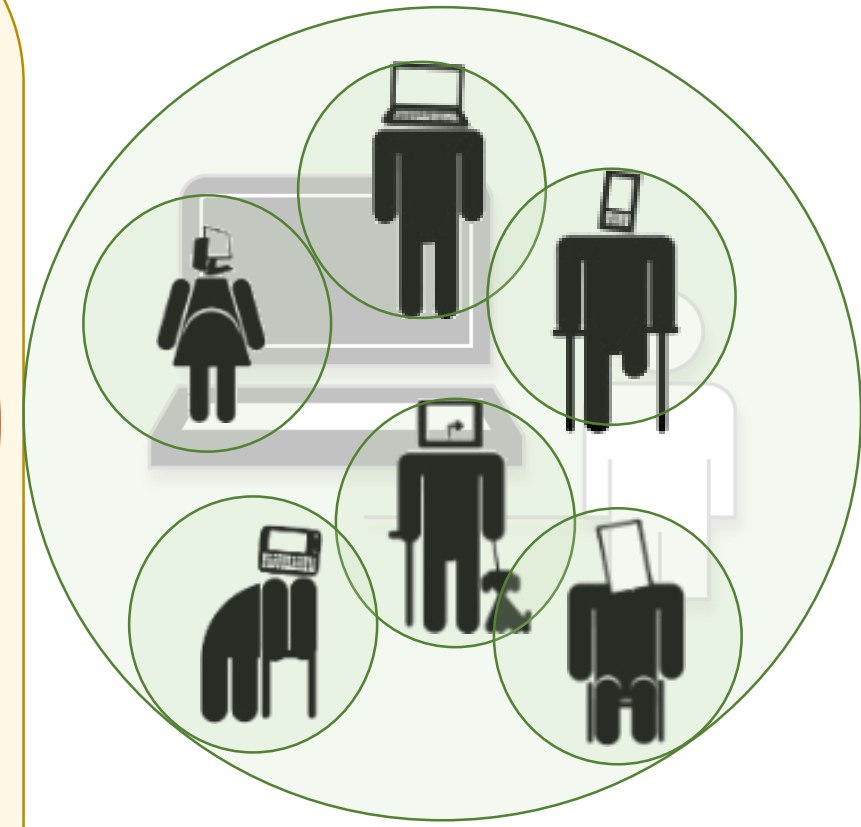
Game Logic



Game View



User

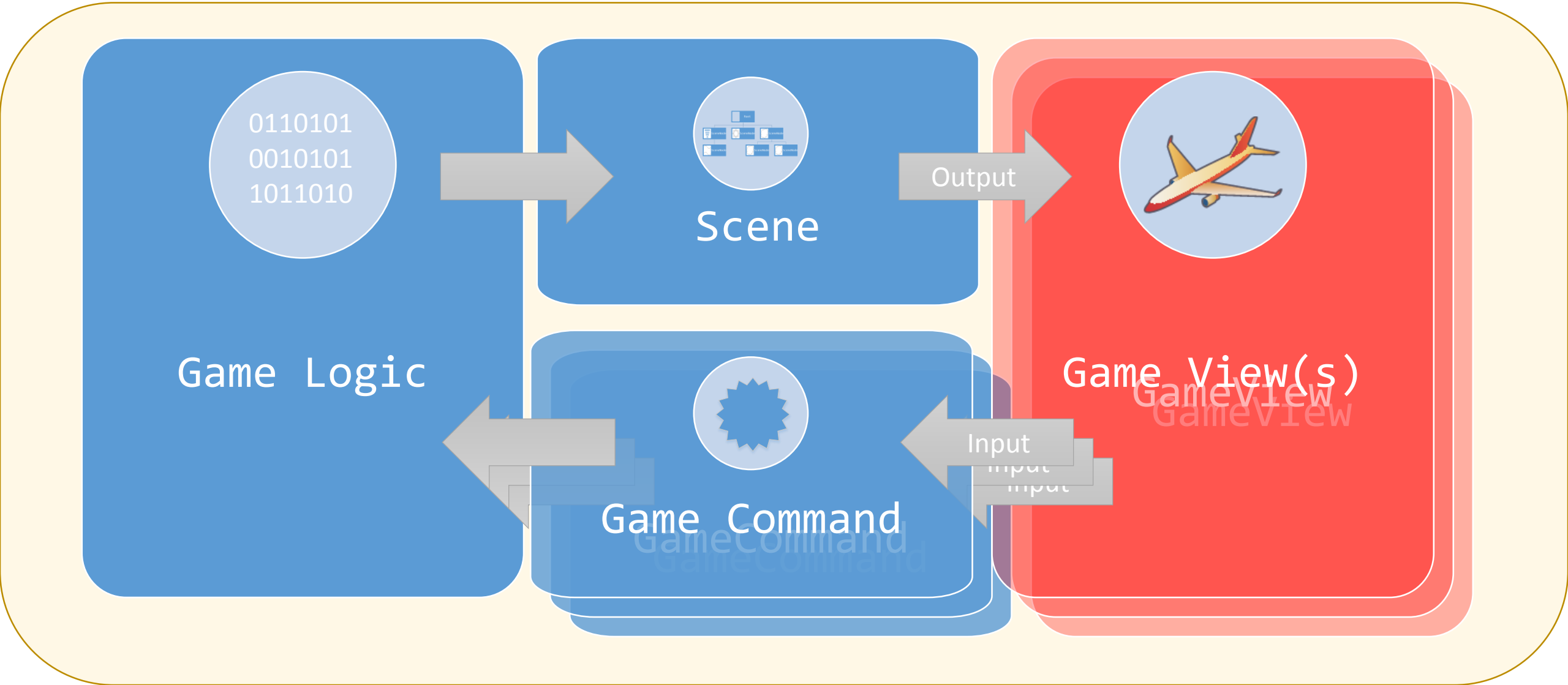


Game Application

The Game Application

- The Game Logic and all the defined Game Views and Game Controllers are part of the Game Application.
  - The Game Application allows defining:
    - The desired output subsystem (for instance, OGRE and OpenAL Soft);
    - The desired input subsystem (for instance, OIS);
    - The desired scripting subsystem (Lua in progress).
- The Game Logic might be the same. However, the Application's Game View might vary greatly.
- The Game Application may provide different player profiles.
  - Some profiles might provide different games settings for average users;
  - Others may target different disabilities.

The Game Application



Next Steps

- We kindly ask for your help to improve UGE.
  - We are conducing an evaluation at: <https://github.com/francogarcia/uge-evaluation>.
- A sample prototype for a UA-Game exploring the described approach is bundled with UGE's source code.
  - The code is available at: https://github.com/francogarcia/uge ;
  - It is a simple, quick prototype. However, it illustrates UGE's run-time tailoring potential.
- The documentation describes an step by step tutorial to implement the prototype. This prototype is defined as suggest in this guide: with an IO-free Game Logic and player profiles to specialize the game to different interaction needs.
  - The Developer's Reference is available at: <https://github.com/francogarcia/uge/raw/master/doc/Documentation.docx>.