

Video Game Smells

Video Game Smells

We are a team of researchers from Polytechnique Montréal (Canada) and the University of Sannio (Italy) that are conducting research on bad practices in video game development. In order to better finalize our study we are proposing a survey, we would like to invite you to participate in it. We would be grateful for that. Personal data will be treated as strictly confidential.

The goal of this survey is to assess a catalog of smells concerning Video Game development. The catalog is organized into 5 sections, concerning different categories of problems. For each question, you need to provide an answer indicating whether or not you agree on the relevance of the reported smell.

Participation in this survey is voluntary and confidential, it is expected to take about 20 minutes of your time, you might withdraw at any time.

Note: You can assign a number of stars to each elicited smell highlighting whether it describes a highly critical problem or a less critical one. A highly relevant problem is one that in your experience seriously impacts the development and maintenance of your video game, in terms of resources spent and achieved efficiency. At the end of all categories, you can suggest any other kind of smell not mentioned in our catalog. We can use it to improve our study. If in your opinion the smell under analysis does not impact negatively/positively your video game code, you can use three stars. Moreover, if you do not have enough knowledge of the reported smell, you can leave it blank.

We thank you in advance for your collaboration. We will only analyze aggregated data, and individual responses will not be traced back to an individual respondent.

If you have questions about the questionnaire or our research, please do not hesitate to contact us.

Vittoria Nardone (vittoria.nardone@unisannio.it)

Biruk Asmare, Muse (biruk-asmare.muse@polymtl.ca)

Foutse Khomh (foutse.khomh@polymtl.ca)

Massimiliano Di Penta (dipenta@unisannio.it)

Design and Game Logic

This category groups bad practices due to bad design choices and game logic problems.

Search by string/ID

Searching objects by tags/strings (e.g., in Unity through GetComponent) or repeatedly comparing objects' tags may cause performance degradation



Comments (Optional)

Creating components/objects at run-time

Repeatedly creating/destroying game objects at run-time may cause performance issues. It is preferable to use an object pool when many objects are instantiated and disposed of very frequently.



Comments (Optional)

Lack of separation of concern

Scripts/classes are incohesive, mixing up controller handling, physics, animation, and rendering.



Comments (Optional)

Prefer static classes instead of singleton for single instance entities in the game

Singletons, differently from static classes, can be derived from existing classes. Also, their usage is transparent to clients (as opposed to static classes). At the same time, when designing singleton care should be taken to avoid making it non-thread-safe.
See "Design Patterns: Elements of Reusable Object-Oriented Software" written by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides



Comments (Optional)

Dependencies between objects

References to game objects of game object components are (unnecessarily) repeatedly retrieved



Comments (Optional)

Controller inputs separated from game logic

Input handling from controllers (e.g., joystick, keyboard) and game logic decoupled



Comments (Optional)

Poor design of object state management

The way a game object state is handled is poorly designed. For example, complex state management might suggest using a state design pattern instead of an if structure. Another problem is when the state of a game object is handled by an external component different from the game object itself, creating unnecessary coupling



Comments (Optional)

Static coupling

In some game engines (e.g., Unity) dependencies between objects are set statically through the IDE inspector. This is easy to do, but may cause maintainability problems in the source code as dependencies are not visible there



Comments (Optional)

Bloated asset

A redistributable asset (e.g., a model is bloated of useless and heavyweight components, e.g., unnecessary textures, animations, etc.)

Not
Critical

Highly
Critical



Comments (Optional)

Weak temporization strategy

There is a weak temporization in the game (e.g., physics simulation depends on the frame rate, or there is a lack of sync between the updates of multiple game objects)

Not
Critical

Highly
Critical



Comments (Optional)

Multiplayer

This category groups bad practices leading to multiplayer problems.

Status stored on clients for multiplayer

A client can modify the status of another player's object, making possible attacks during games (e.g., preventing the competitor to play/win the game)



Comments (Optional)

Inefficient data transfer between client and host/servers

In a multiplayer game, there is an unnecessary (and verbose) transfer of data between different peers



Comments (Optional)

Reload entire workspace when a new client connects

Every time a new client connects to a multiplayer game, the entire scene is reloaded causing lags/performance problems to all players



Comments (Optional)

Fail to transfer game objects

The client does not transfer the whole scene to other clients at every update



Comments (Optional)

Syncing multiple textures in a multiplayer environment

Drawing and syncing many canvas/dynamic textures over clients/servers of multiplayer games causes lags or even out-of-sync glitches. The solution is to simplify the number of canvas/textures to draw and sync



Comments (Optional)

Animation

This category groups bad practices leading to animation problems.

Improper use of rigging

There is an improper use of rigging in a composite object/model



Comments (Optional)

Multiple animators over a model components

A game object uses (unnecessarily) multiple animator components (or in general components responsible to handle the animation of the game object, depending on the technology being used). A single animator, even with a complex state machine governing the animation, would make things much clearer



Comments (Optional)

Moving player vs. moving the rest of the scene in an endless-world like game

The choice between moving the player game object vs. moving the rest of the scene has not been made properly. In some games, it may be convenient to move the scene (e.g., endless run) and not the player, or vice versa



Comments (Optional)

Continuously checking if position/rotation is within the boundary

When animating/moving an object, the game continuously checks whether the object is within a boundary (other mechanisms such as range clamping or colliders could be used instead)



Comments (Optional)

Use objects over particle systems when not needed

The game uses objects to create effects that can be achieved using particle systems instead



Comments (Optional)

Animation state handling kept separate

Animation state handling is kept decoupled from the animations themselves



Comments (Optional)

Too many keyframes in animation

An animation contains too many keyframes, and this causes performance issues



Comments (Optional)

Physics

This category groups bad practices leading to elements physics.

Improper mesh settings for collider

There is a suboptimal choice of collider for an object (e.g., too expensive mesh collider, or too-coarse collider instead)



Comments (Optional)

Heavy physics computation in update

The games perform heavyweight physics computation at every update



Comments (Optional)

Setting objects velocity and override forces

When this is not necessary, objects' velocity is directly modified, instead of operating through Forces/Physics



Comments (Optional)

Physics computation kept separate

Physics is computed before handling animations



Comments (Optional)

Rendering

This category groups bad practices leading to rendering problems.

Object drawing/rendering not properly optimized

The way objects are drawn is not optimized. For example, too far, not visible objects are always redrawn, or all objects of a scene are redrawn at every frame (and not just those that have been changed)



Comments (Optional)

Excessive number of elements (meshes, textures, sprites) in the scene

Using an excessive number of elements leads to performance issue, (e.g. use an excessive number of meshes instead of combined ones)



Comments (Optional)

Texture or polygon choices cause aliasing

A bad choice of object texturing or number/type of polygons of a game object causes aliasing



Comments (Optional)

Sampling and Rendering to the Same Texture

Writing to the same texture and then rendering it can cause undefined behavior in the GPU. Instead, a texture copy should be created and modified



Comments (Optional)

Inefficient rendering of objects

The way objects are rendered is inefficient (e.g., related objects are rendered one-by-one)



Comments (Optional)

Suboptimal, expensive choice of lights, shadows, or reflection.

A suboptimal choice of lights, reflections, or shadows may cause performance problems. For example, this happens when some lights that can be baked are, instead, rendered in realtime, or when there is excessive usage of (unnecessary) shadows and reflections

Not
Critical

Highly
Critical



Comments (Optional)

Issues in material rendering

The material rendering (e.g., wrong mesh type, two-side materials) causes visualization issues

Not
Critical

Highly
Critical



Comments (Optional)

Would you like to suggest any other kind of smell not mentioned in the previous sections?

Demographics

Name, Surname

Organization

Which is your educational qualification?

☐

High school

☐

Bachelor Degree

☐

Master Degree

☐

PhD

☐

Other

Which is your role in your current organization?

☐

Software Developer

☐

Product Owner

☐

Technical Manager

☐

Computer Graphics

☐

Other

How many years of experience do you have in video game development?

How many years of experience do you have in software development?

Which languages do you use to develop video games? *

You can select multiple options.

☐ C/C++

☐ Java/JavaScript

☐ Python

☐ C#

☐ PHP

☐

Which game engine/framework do you use to develop video games? *

You can select multiple options.

☐ Blender

☐ CryEngine

☐ Unity

☐ Unreal Engine

☐

Would you like to be available for an interview?

☐ Yes

☐ No

If Yes, please enter your email address:

@ |