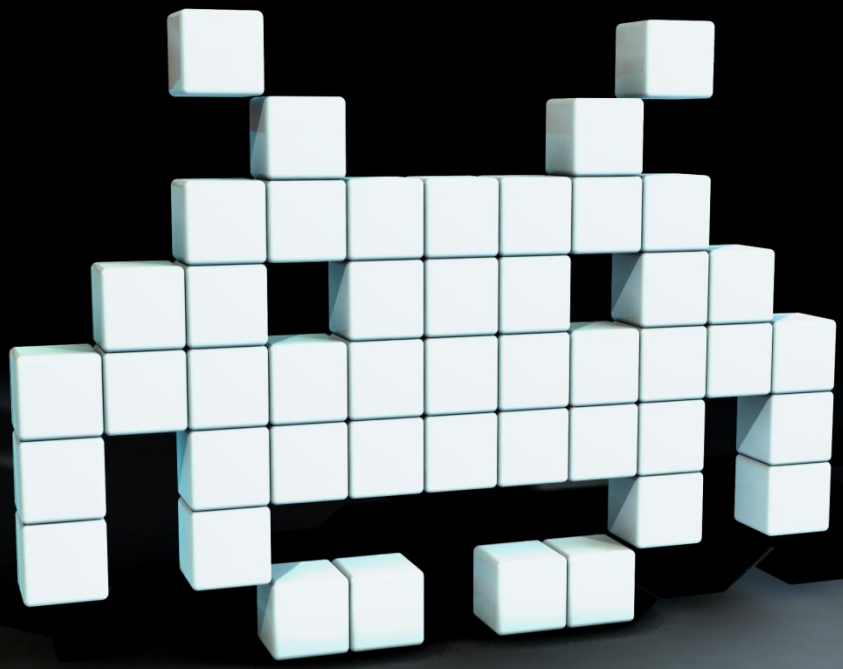


Summer Games University

Day 2: Core Mechanics

Hell, it's about time!

Image source: "Starcraft 2" Blizzard Entertainment

Synchrony

Why think about synchrony of operations?

Why is multiplayer difficult?

Image sources: "Sins of a Solar Empire" Ironclad Games, "Aion" NCsoft

Synchrony

Players should share the same world state

Player interaction has to be handled (and probably analyzed)

Real-time capable ping times

What could cause asynchrony?

Different Cycle resolutions, network lag, different order on updating Game Objects, float calculations, hardware, cheating, random variables, circular causal dependencies between Game Objects, different game settings, multithreading vs measuring time, uninitialized variables, bugs, ...

Synchrony

What do we actually need to sync?

- Game Object states
- Changes on Game Object states
- Input

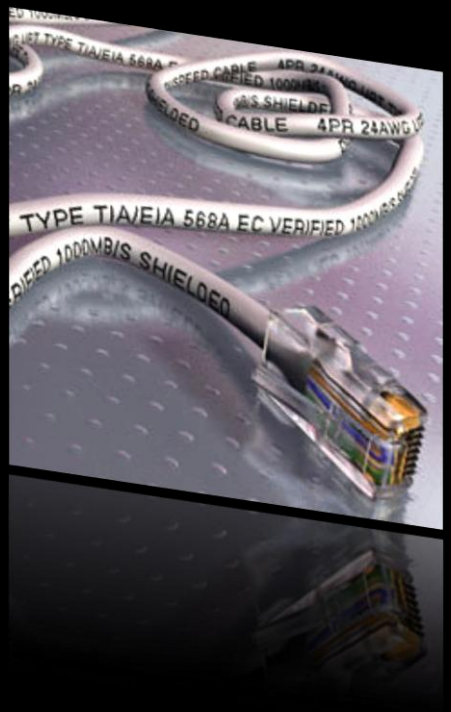
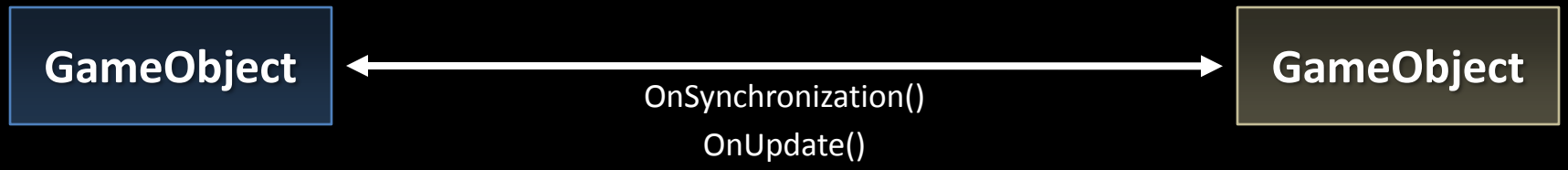


Image source: Turbosquid

Synchrony on Game Object level

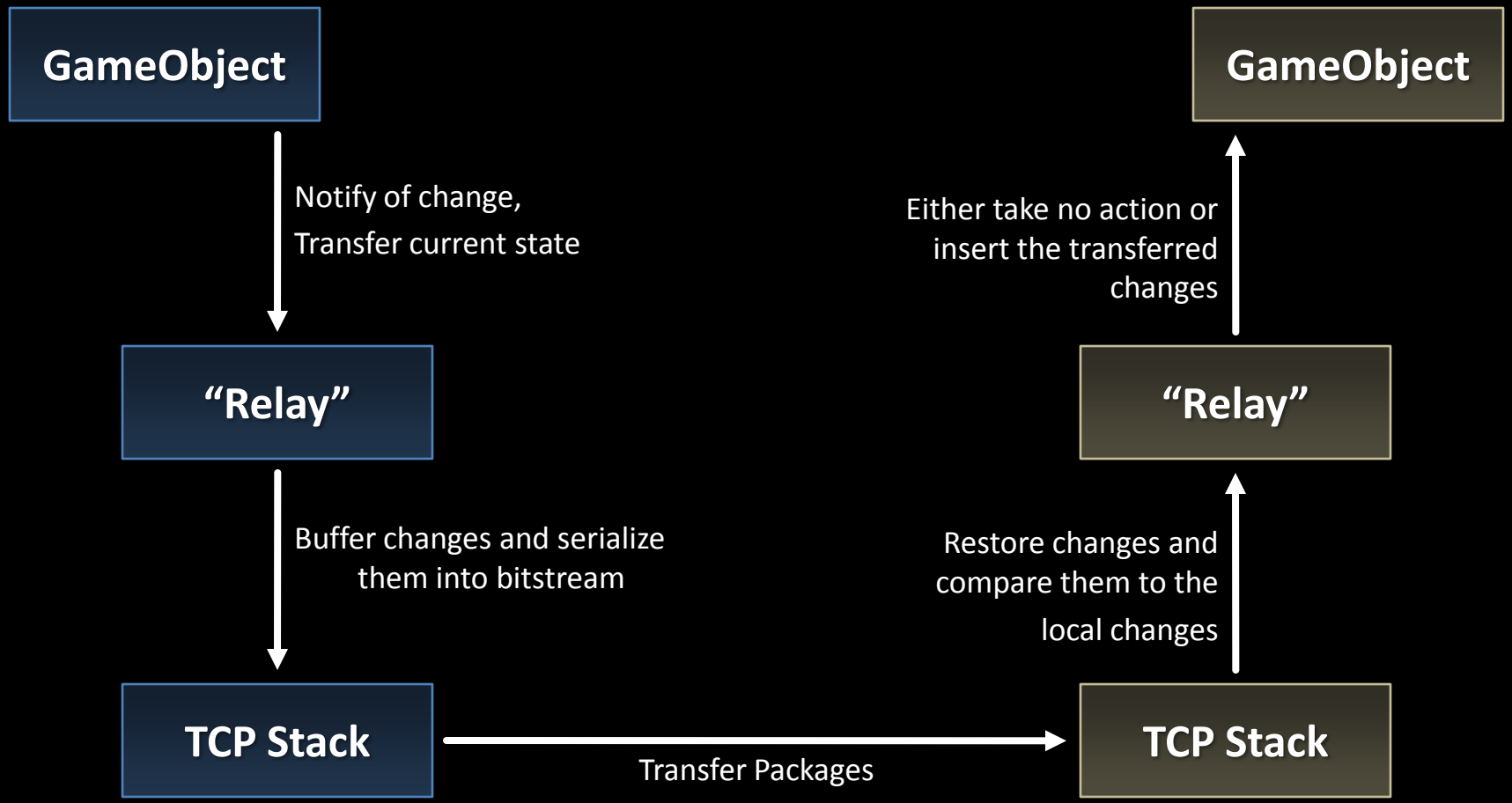
Lets start with synchronizing Game Object states



There are different synchronization patterns, deciding which client will calculate what:

- If the complete Game Object information is transferred, its only necessary to calculate the updates on one client.
- In most games, all clients calculate their changes and they are checked for asynchrony.

Synchrony on Game Object level



Synchrony on Game Object level

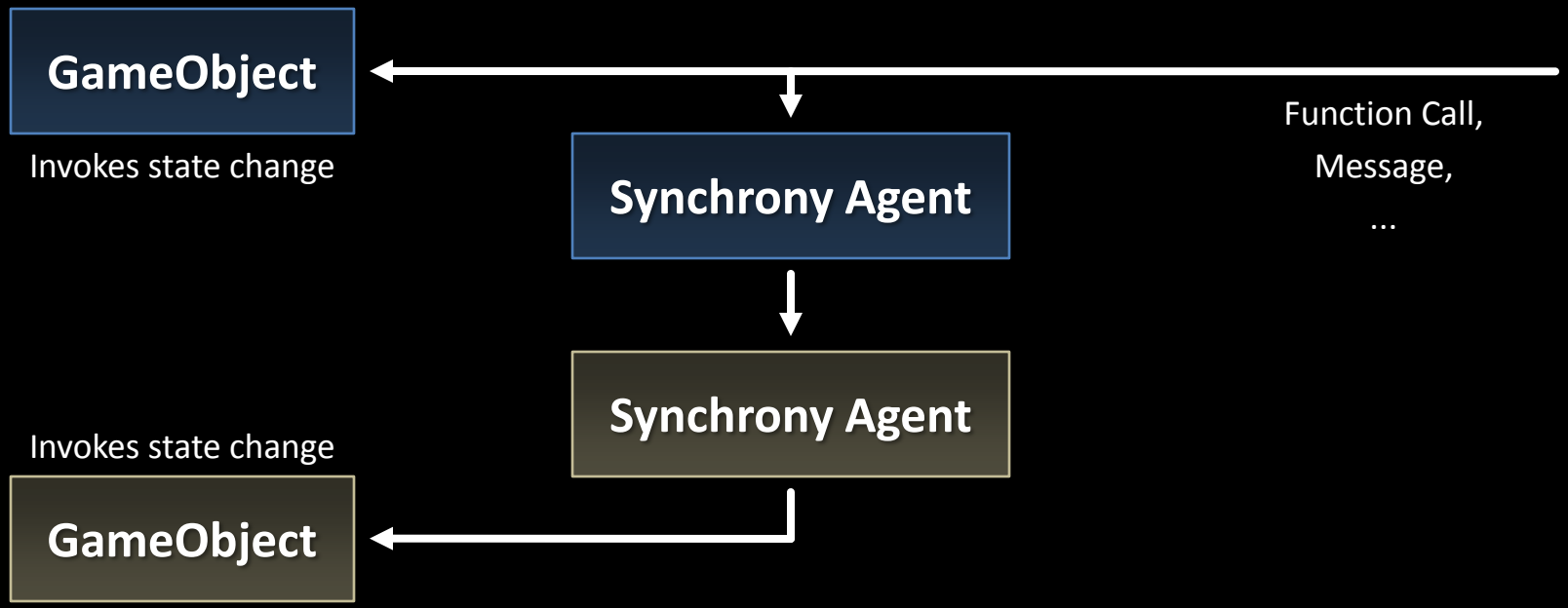
The single Game Object update steps are often wrapped in a “Cycle Report”

The Cycle Report is an incredible good pattern to use when starting development on a first engine, as it is an easy method to achieve both parallelism and multiplayer synchrony.

Synchrony on update level

In some game scenarios, transferring the state updates leads to a lot of overhead. For example if complex game action scripts are used.

Let's try to reduce the amount of data to transfer by taking a look where the actual state updates came from.



Synchrony on update level

While this solution sounds straight forward, it will cause some serious constraints we will have to follow. The most important (and difficult one):

All clients have to produce the same state for a given update notification!

Update Message: Accelerate(vector3(1, 0, 0))

```
velocity = velocity + arg0
```

All fine!

Update Message: Accelerate(vector3(0.2005293318, 0, 0))

```
velocity = velocity + arg0
```

ASYNC!

Synchrony on update level

Enforcing these extra constraints might cause processing overhead due to additional checks or more precise data types. Random() is another popular example.

You might also have to scrap all third party software, as they usually don't follow these constraints (physics engines).

It's all about "Is it worth the effort?" again.

Image source: Life, coffee & the pursuit of ...

Synchrony on input level

Take it another step further: Where do the update notifications come from?

We could also just transfer a players input and produce the update notifications and the state changes on all clients.

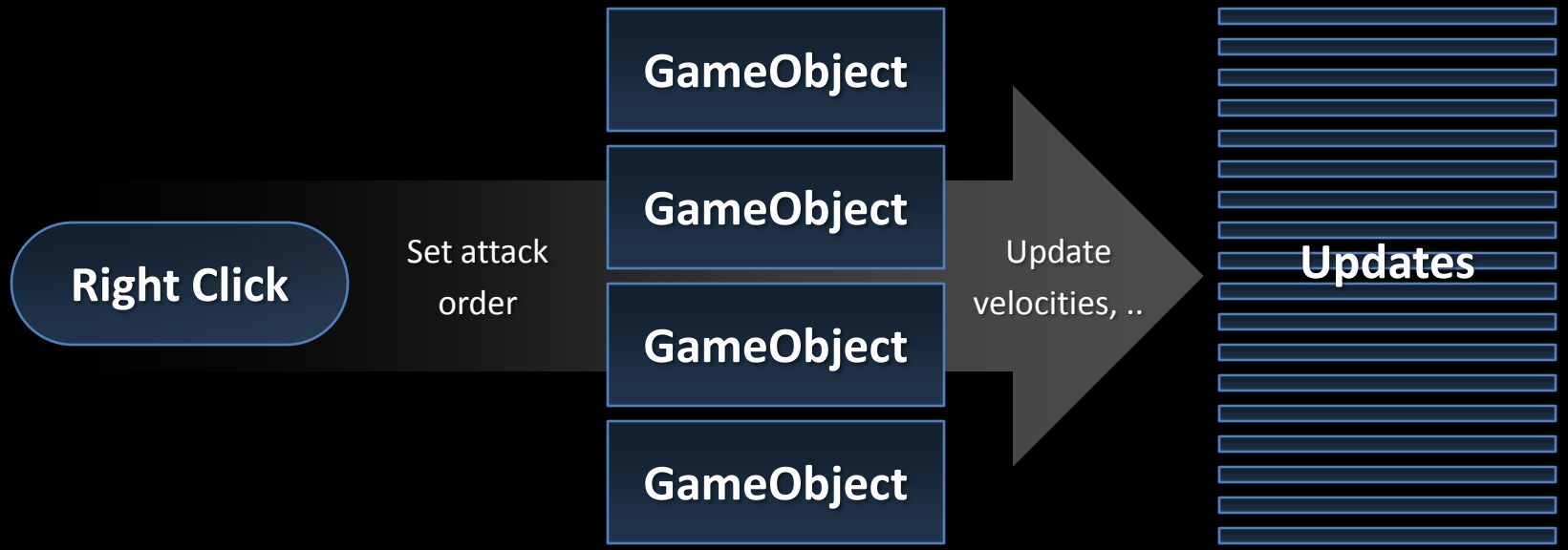
Adds a ***DAMN HELL*** of extra constraints to care for. In a generic case we can't even process the Game Objects in any order..

Image source: "Supreme Commander" Gas Powered Games

Synchrony on input level

In most genres this pattern will only lengthen your development time and might even reduce performance. The player's input on a generic shooter will match quite closely on update notifications or even state changes.

Usually only used in Real Time Strategy games - the most popular examples being StarCraft (1) and Supreme Commander.



Game Objects and the Universe – Part 2

How to deal with our Game Objects?

And how to ***speed*** it up!

Image source: NASA

Game Objects and the Universe – Part 2

Okay, so let's continue working with a prime cycle to do the update job.

How should Game Objects controlled by a cycle interact?

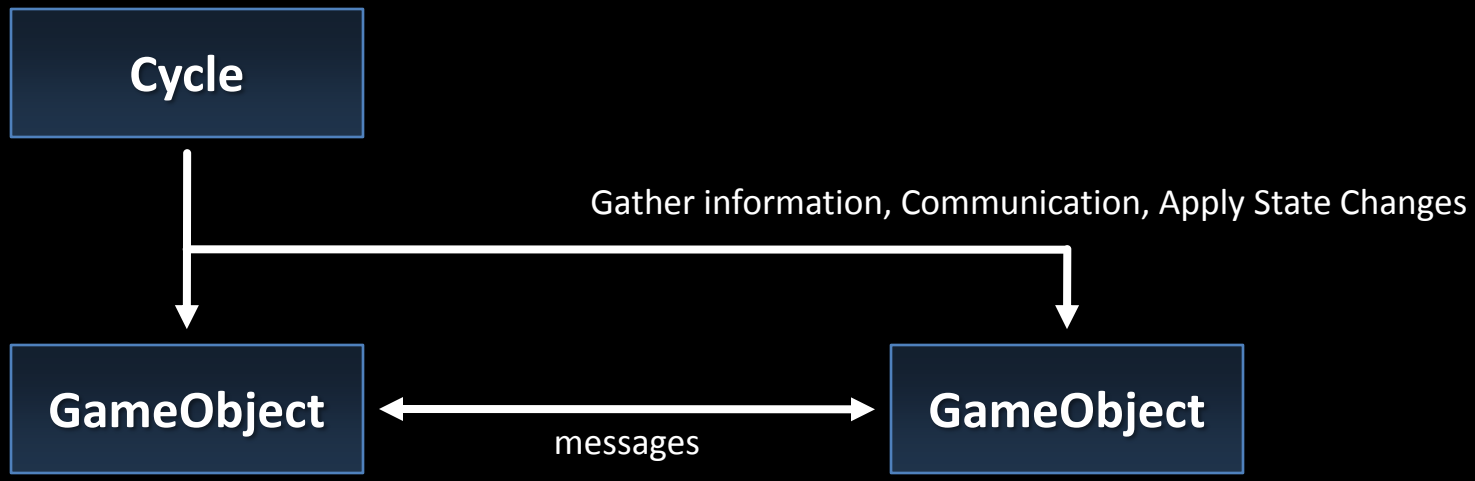
And don't forget multi-threading!

Image source: Shaoxing Changsheng Thread Manufacturing Co., Ltd.

Game Objects and the Universe – Part 2

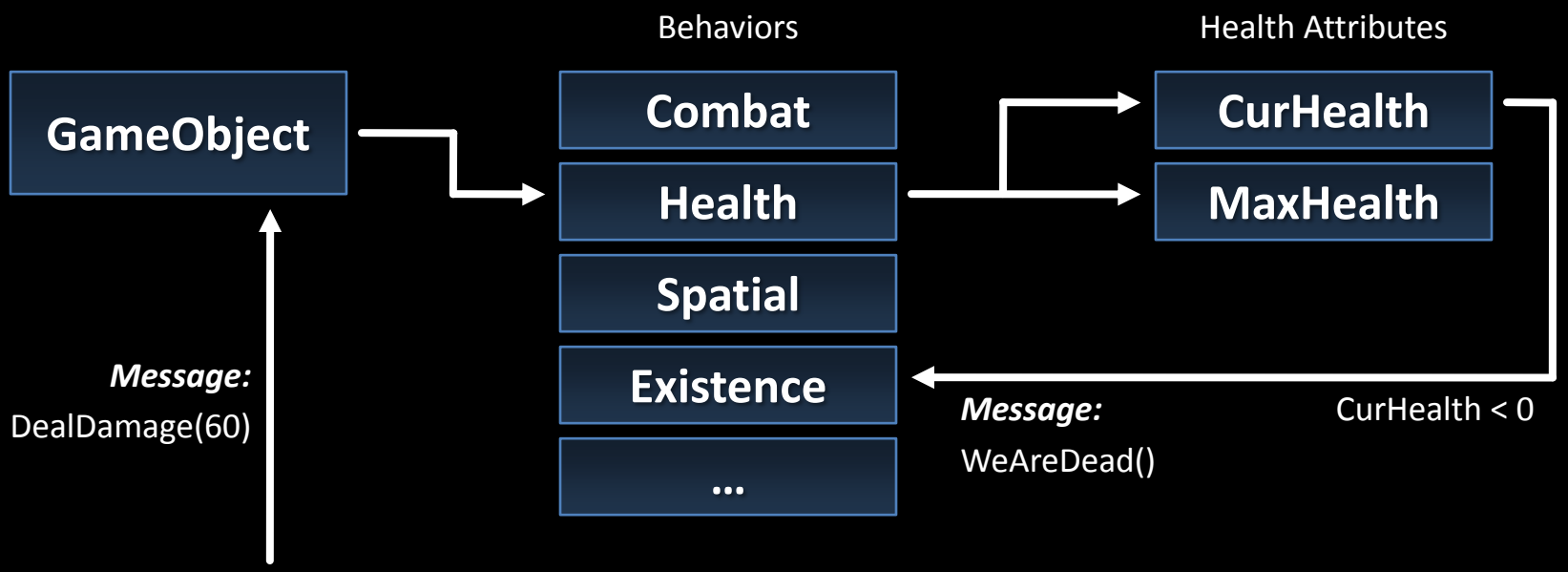
Game Objects might be allowed to interact freely at given points of time

Messages are used to solve causal dependencies



Game Objects and the Universe – Part 2

A good Game Object representation for a message based system is called Behaviors. Game Objects are nothing else but Behavior containers.



Game Objects and the Universe – Part 2

Behaviors involve a performance loss, as a lot of message redirection is involved.

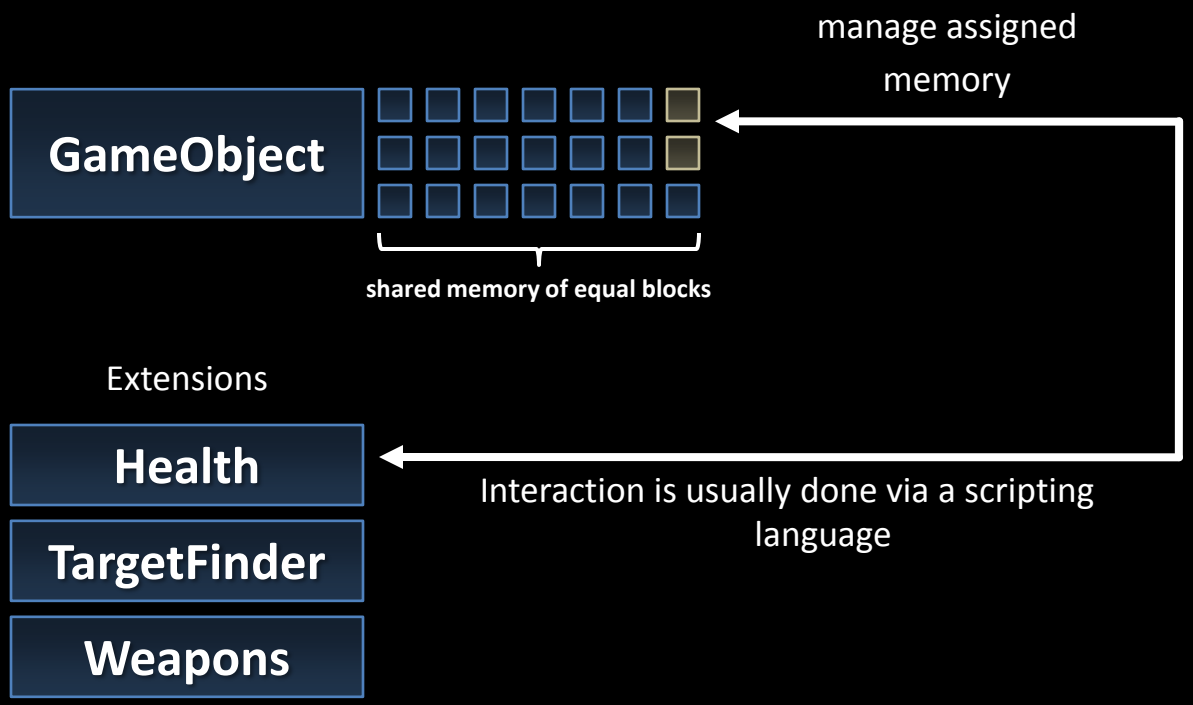
There are still some beneficial features:

- Behaviors offer a tremendous capability to reuse Behaviors for different types of Game Objects. That's possible because all Behaviors communicate through defined message channels
- Intercepting the messages gives us a perfect foundation to integrate an Synchrony on update level NetCode

Due to this characteristics, Behaviors are best suited for shooters or similar genres.

Game Objects and the Universe – Part 2

Another popular approach is to work with a shared memory block and modular extensions. Game Objects host and control extensions and supply shared memory. Extensions communicate via function calls.



Game Objects and the Universe – Part 2

Shared memory blocks are a straight forward way to accumulate information and functionality in Game Objects and still maintaining a certain degree to reuse code. It's basically a better performing (in terms of real time applications) way to map inheritance.

It is usually utilized if..

- The engine includes a functional scripting language, as the shared memory blocks can easily be mapped on functional atoms like Lambdas
- The game concept does not require a large amount of different game object types

Shared memory blocks are well suited for strategy games or similar genres.

Now its getting fun..

physics!

Image source: Funny Pictures

Physics

The simulation of realistic physics has become an important aspect in modern games. Worth taking a closer look onto them!

Very much like game engines, a physics engine simulates its own time, causality and object space.

Let’s focus on rigid bodies for now.

Physics

The main job of a (rigid body) physics engine is to simulate collisions.

Forces	Apply forces (or something similar) to physics objects to transform them.
Collision Detection	Detect if (and if so, where) physics objects collide. Then starts a Collision Response.
Collision Response	Change the movement and rotation of colliding physics objects to create the illusion of solid mass.

Physics Movement

Moving objects without collision is relatively trivial. Sadly, this almost only happens in space.

Physics engines do usually include friction dampening for movement transformations to simulate atmospheres.

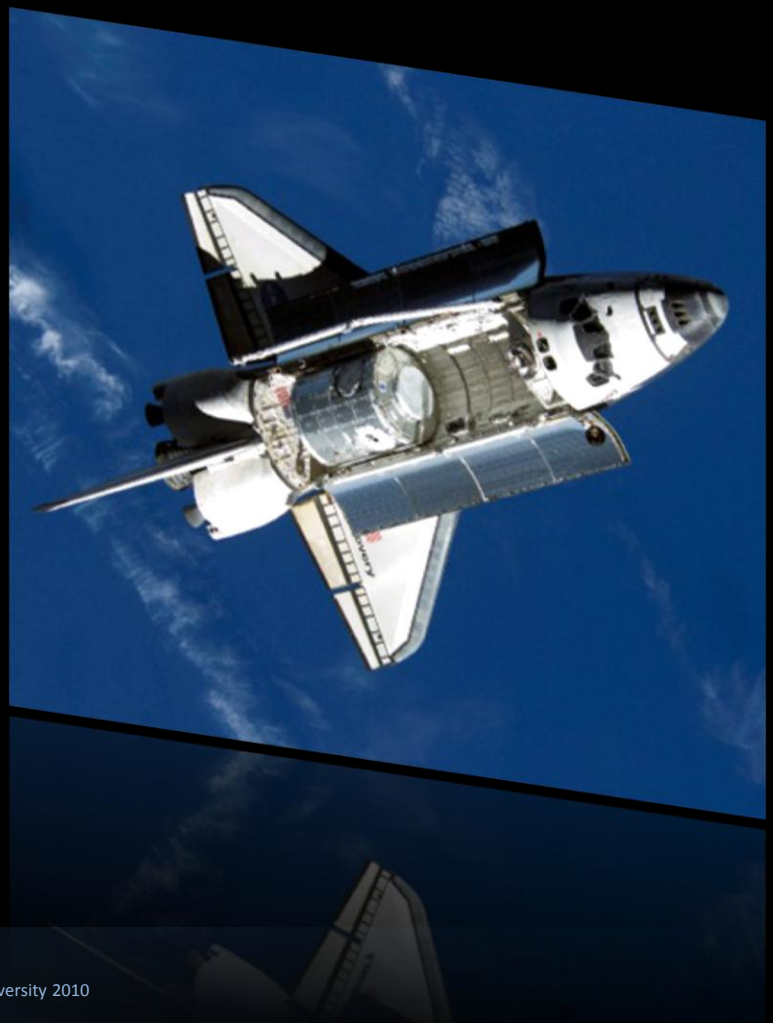


Image source: NASA

Physics Collision Detection

Now it’s getting tricky. We will focus on Collision Detection of rigid bodies for now. I’m open to a “Physics Day” ☺

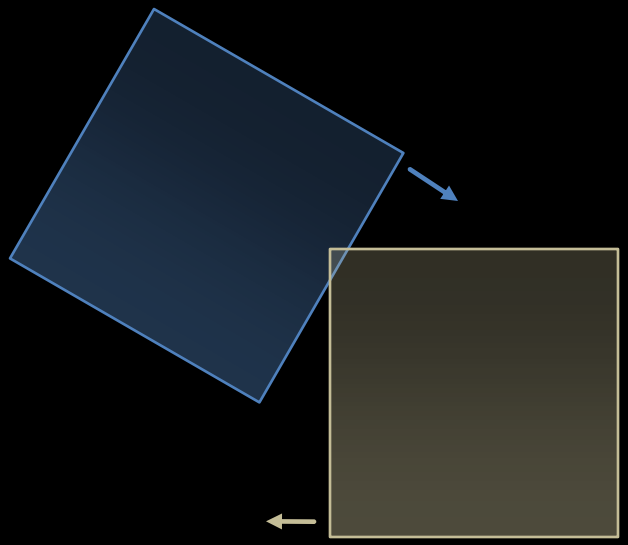
Physics engines check for interpenetration rather then collision and then try to construct the last state where bodies did not intersect to apply the Collision Response.

Image source: Billiard and Dreams

Physics Collision Detection

The first step to detect a collision within reasonable time is to create physics geometry for your objects, as render geometry is far to complex to be used.

Most physics engines do also require convex geometry, but concave geometry can be modeled by combining multiple convex meshes.



Physics Collision Detection

How to get a fast (and good) collision detection?

The most popular algorithm used for Collision Detection is the Gilbert-Johnson-Keerthi algorithm (or GJK). As you will encounter GJK for sure if reading about physics engines. I will shortly summarize the idea behind it for an easy start.

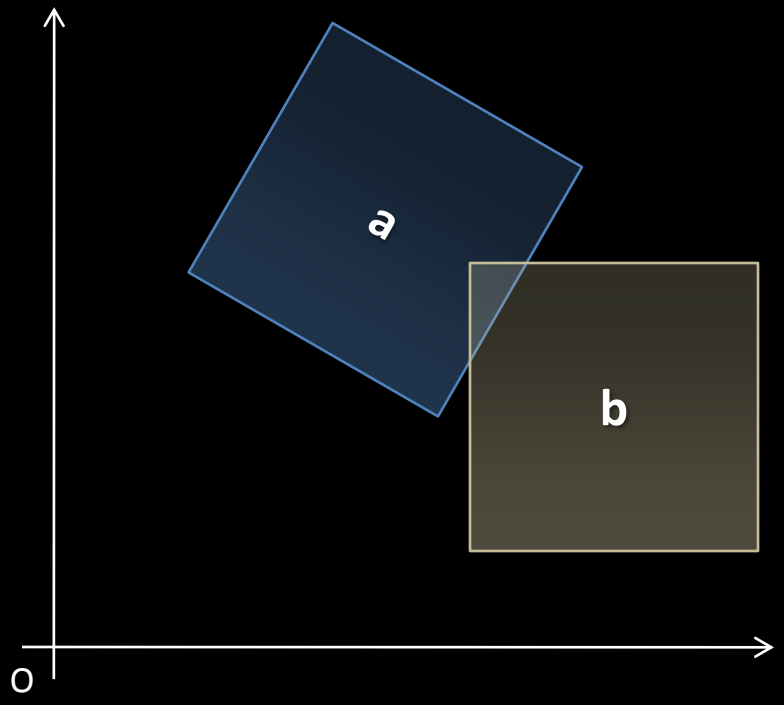
http://www.pfirth.co.uk/minkowski.html

Image source: Firefox

Physics Collision Detection

Minkowski Addition

If the origin is within the Minkowski body, we found a collision



Physics Collision Detection

So how to check if the origin is within a given convex polyhedron?

The idea is to iteratively construct smaller tetrahedrons by starting at any point of the polyhedron and moving “towards the origin”. If we find a tetrahedron which includes the origin, then the polyhedron does also include the origin.

We won’t do that in detail here, but its rather simple once you got the trick. There is a good video explaining what to do to find the tetrahedron (see next slide).

Physics Collision Detection

A quite fast and simple algorithm for ***convex*** shapes.

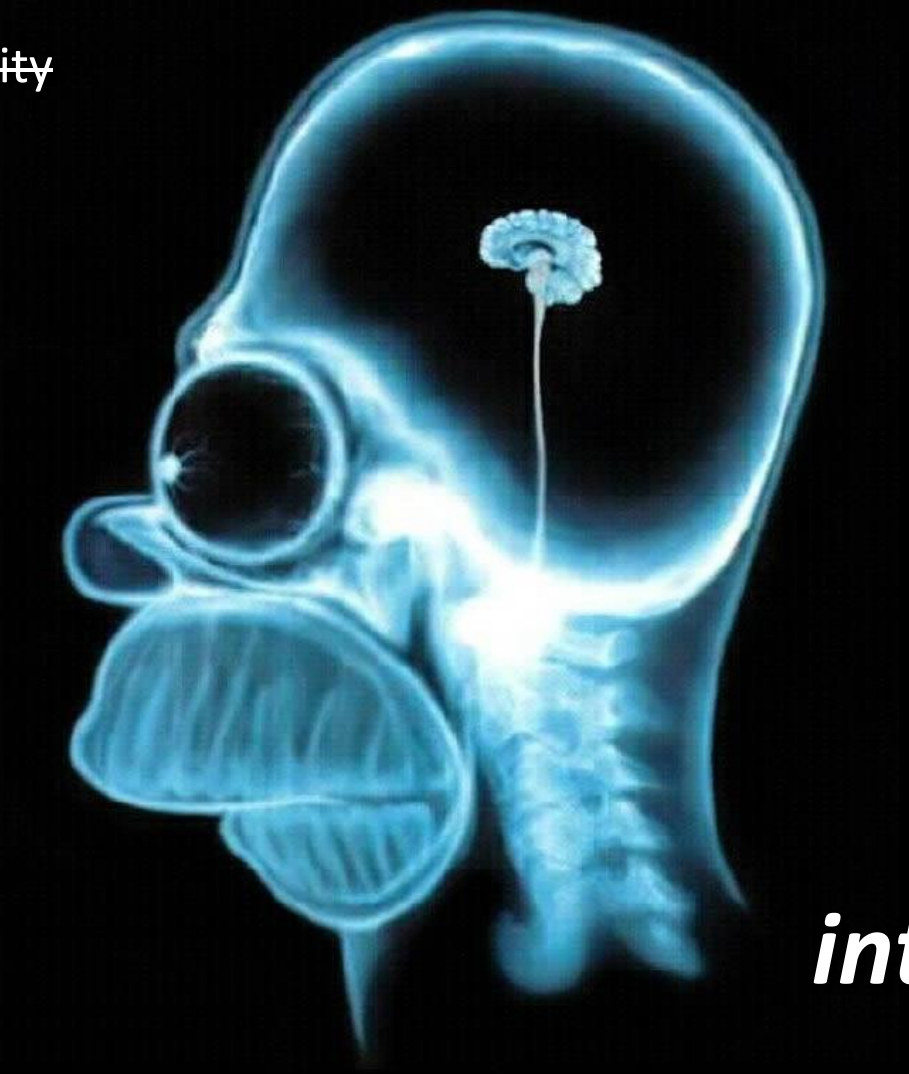
It only returns a Boolean whether or not two bodies intersect. To find the deepest penetration point (which can be used as a collision point and to get the normal of the penetrated surface), there exist several extensions based on the basic GJK.

http://mollyrocket.com/849

Image source: Firefox

Artificial Intelligence

Artificial ~~Stupidity~~



intelligence!

Image source: "The Simpsons" 20th Century Fox

Artificial Intelligence

In the past decades, the graphics where one of the most defining aspects of games – and probably the one with the highest “sell ratio”.
Besides game play of course.

Yet, graphic development is reaching its end!
Whee, I can almost hear the screaming while writing this slide!

A new field will emerge to serve as the “buy me, cause ..” argument!

Artificial Intelligence

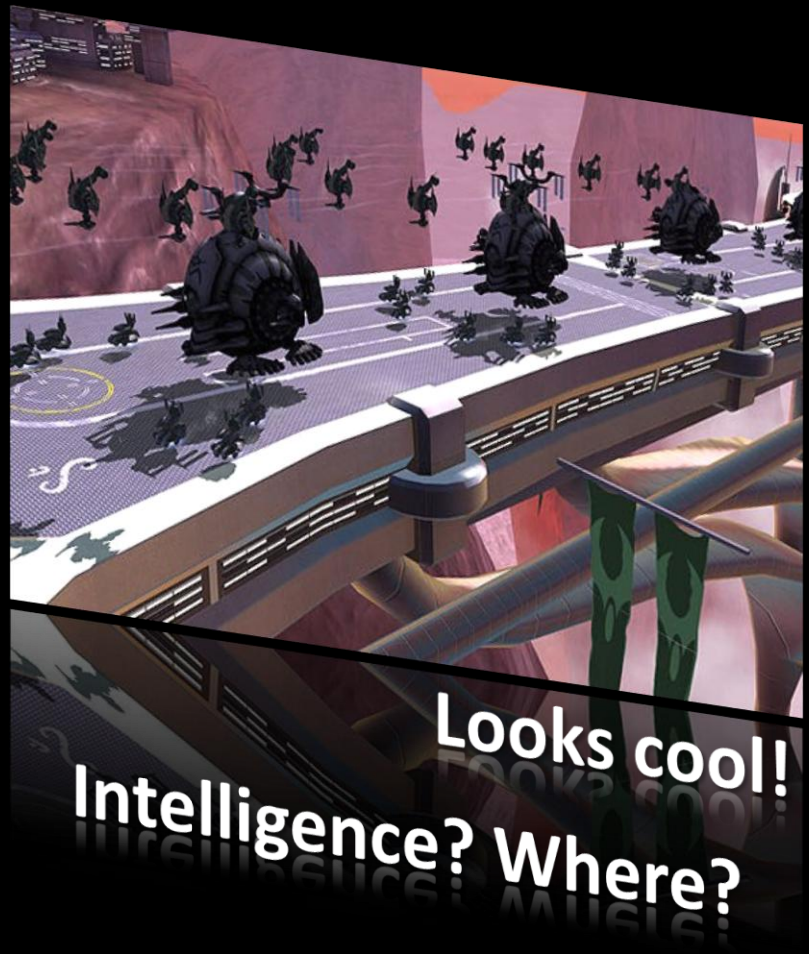


Image source: "Starcraft" Blizzard Entertainment, "Supreme Commander 2" Gas Powered Games

Artificial Intelligence

Okay, so how to integrate a good AI into our engine?

We first have to find out which “metatype” of AI we need

AI methods “Not even recognized”	Pathfinding, Autonomous usage of abilities, ..
Assistance AI “Cool!”	Controlling single entities or parts of entities, units, ..
Player-level AI O_O	Do everything a player would

Artificial Intelligence

Depending on our requirements (especially the required Meta-Type) we have to select a type of AI to use. The most common types are..

Involves decisions made on pure situational parameters without storing any kind of memories

Situational

"A bit of math"

Decisions are made based on limited (usually precompiled) data like scripts, which are altered during the game via memory parameters

Tactical

"Tricky"

Decisions are based on memories which are being accumulated during the game. Look-ahead algorithms are utilized to analyze a potential event path in order to maximize gain

Strategic

"Get me out of here!"

Artificial Intelligence

To make our AI do something, it first has to have a representation of the game universe in some way.

Some games do that by simply adding methods to Game Objects.
That's both dirty and a waste of time!

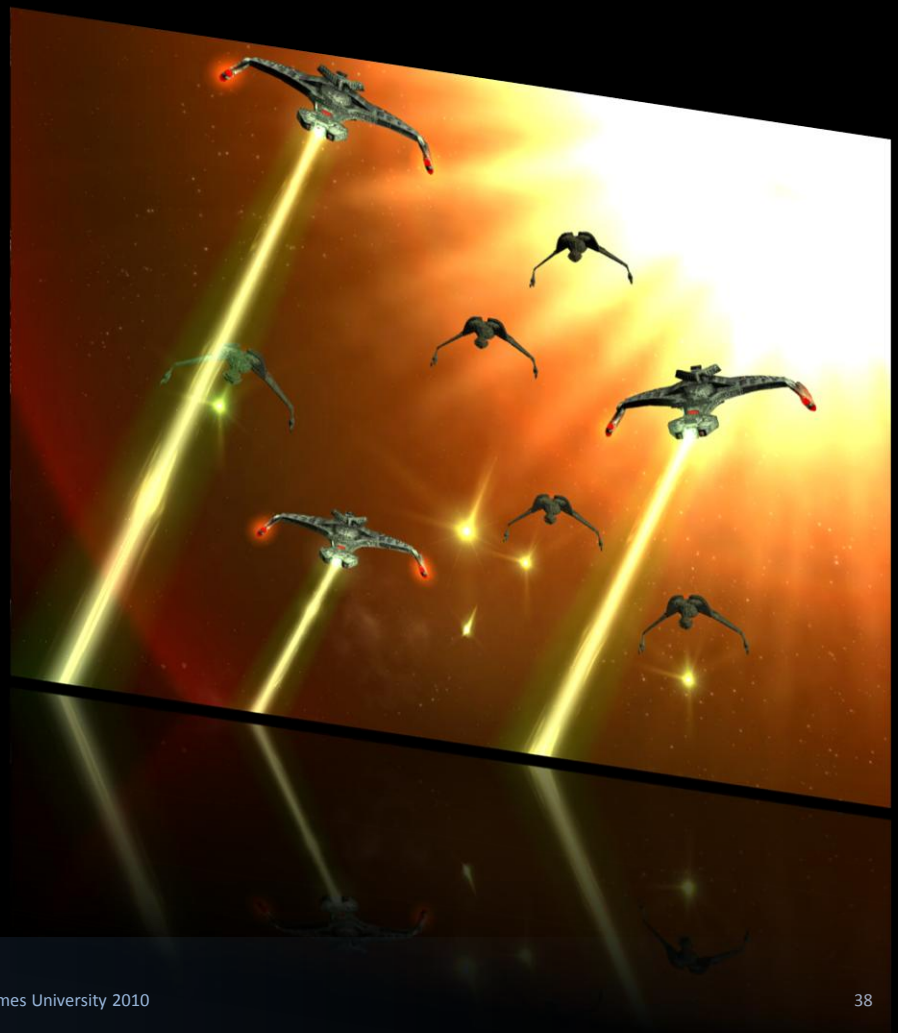


Image source: "Fleet Operations" www.fleetops.net

Artificial Intelligence

A good way to connect a (player-level) AI is to connect it just like you would connect a human player to your universe!

Render something for it!

Of course we won't render pixels, but an AI world that serves as a representation for the AI, to issue commands on game objects (which should be returned just like the commands of another connected human in multiplayer).

This allows us to reuse a lot of architecture, abstract the AI building as far as possible, and keep the AI consistent through content updates (as long as the renderer remains functional, of course).

Artificial Intelligence

Okay, but how to do an strategic AI?

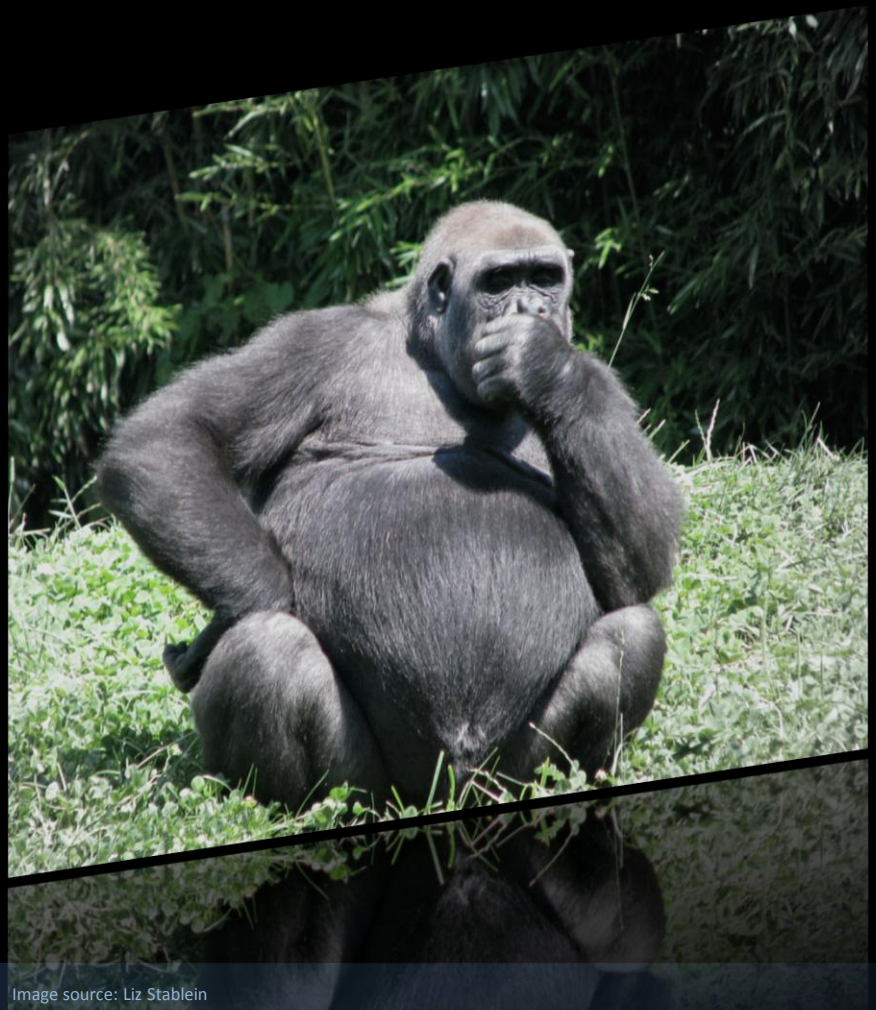


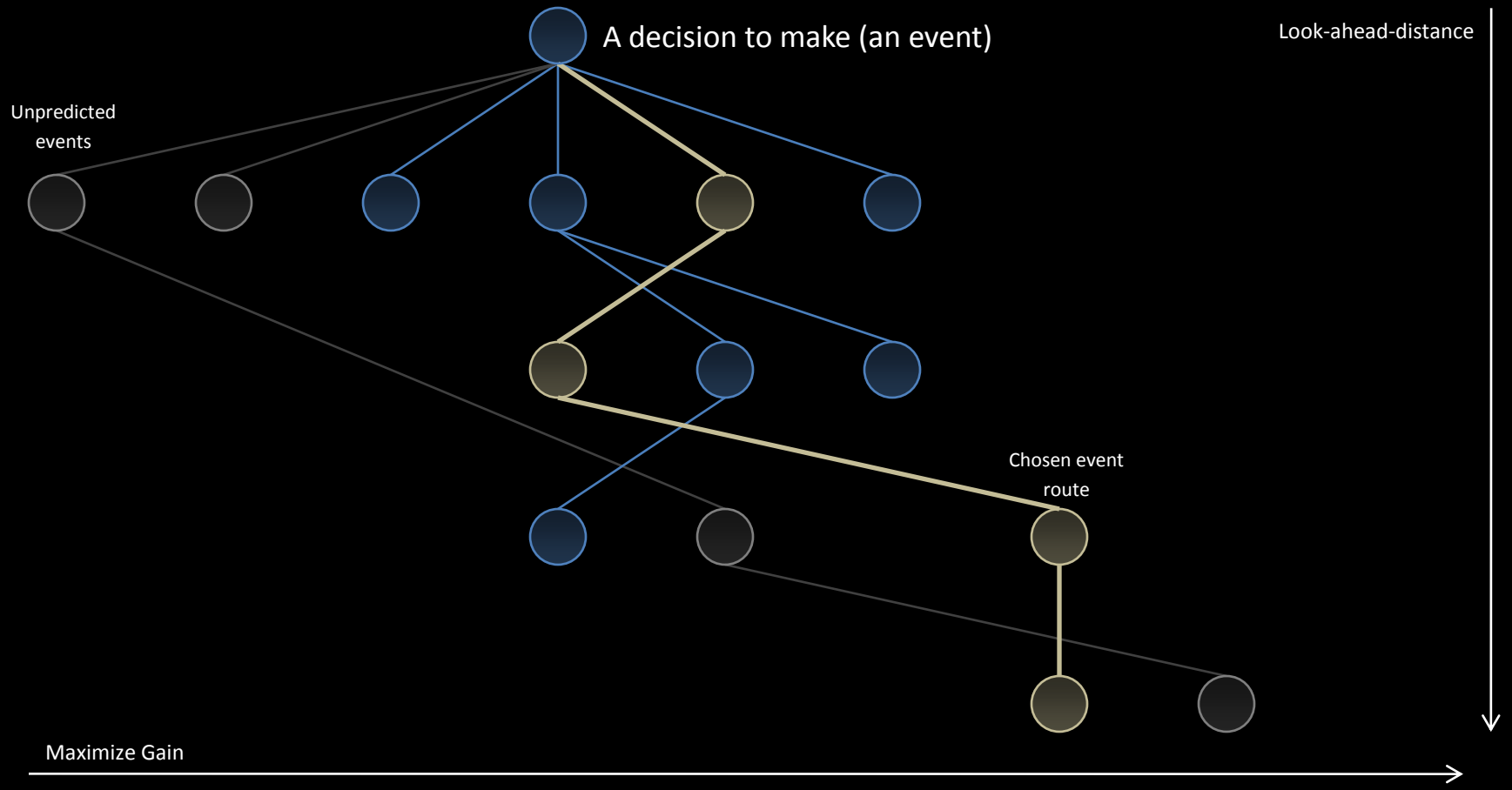
Image source: Liz Stablein

Huu, good question. There are no "real" strategic AIs for real-time applications yet. However, games are indeed one of the most used field of research for AI development.

Let's isolate and try to answer one of the core questions instead: How do we deal with memories?

Artificial Intelligence

Memories are stored as a path, which form a graph of decisions.



Artificial Intelligence

What is stored at each node?

- We should always store the complete chosen path as far as we already evaluated it

- In most cases, its also beneficial to store all potential results for future events which had a larger momentary gain then the decision on this event which we took on our chosen path.
  This is stored to offer fast alternatives, if actions should arise which deny our chosen path or make it worse.

- Its also wise to store additional information, depending on the actual implementation. You could call that “the reasons” why we took the chosen path. Often boils down to a few lightweight parameters.

Artificial Intelligence

Player-level-AI in Real Time Strategy Games.

Clustering AIs

When we are to make a decision, we argue with ourselves what to do. There are several “Players” playing StarCraft at a time within our head.
One tries to maximize resources,
one tries to use this Zerglings as best as possible,
one thinks about which research might be best against the units the opponent is fielding and
one spams the chat with evil emotes. But lets skip the last one for today.

Artificial Intelligence

That’s a good pattern to do RTS AIs too

- Goals
- Economy
- Military
- Buildlist
- ReEval Loop

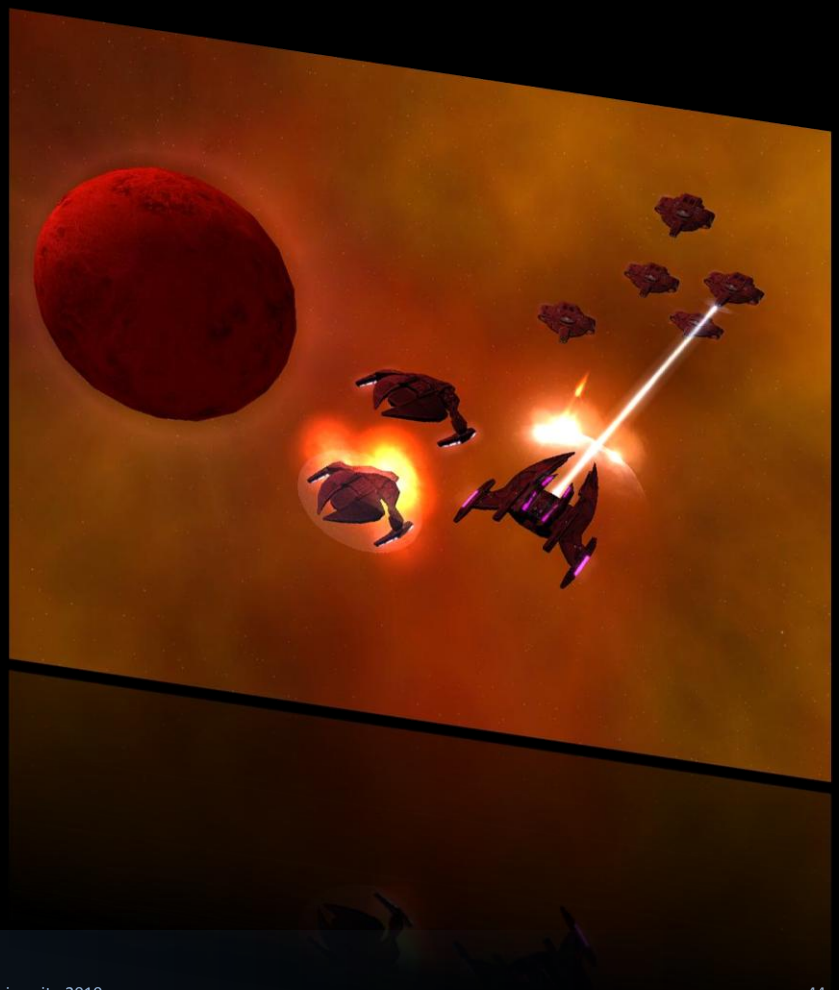


Image source: "Fleet Operations" www.fleetops.net

Connecting the Engine

I don’t want to do all of that!

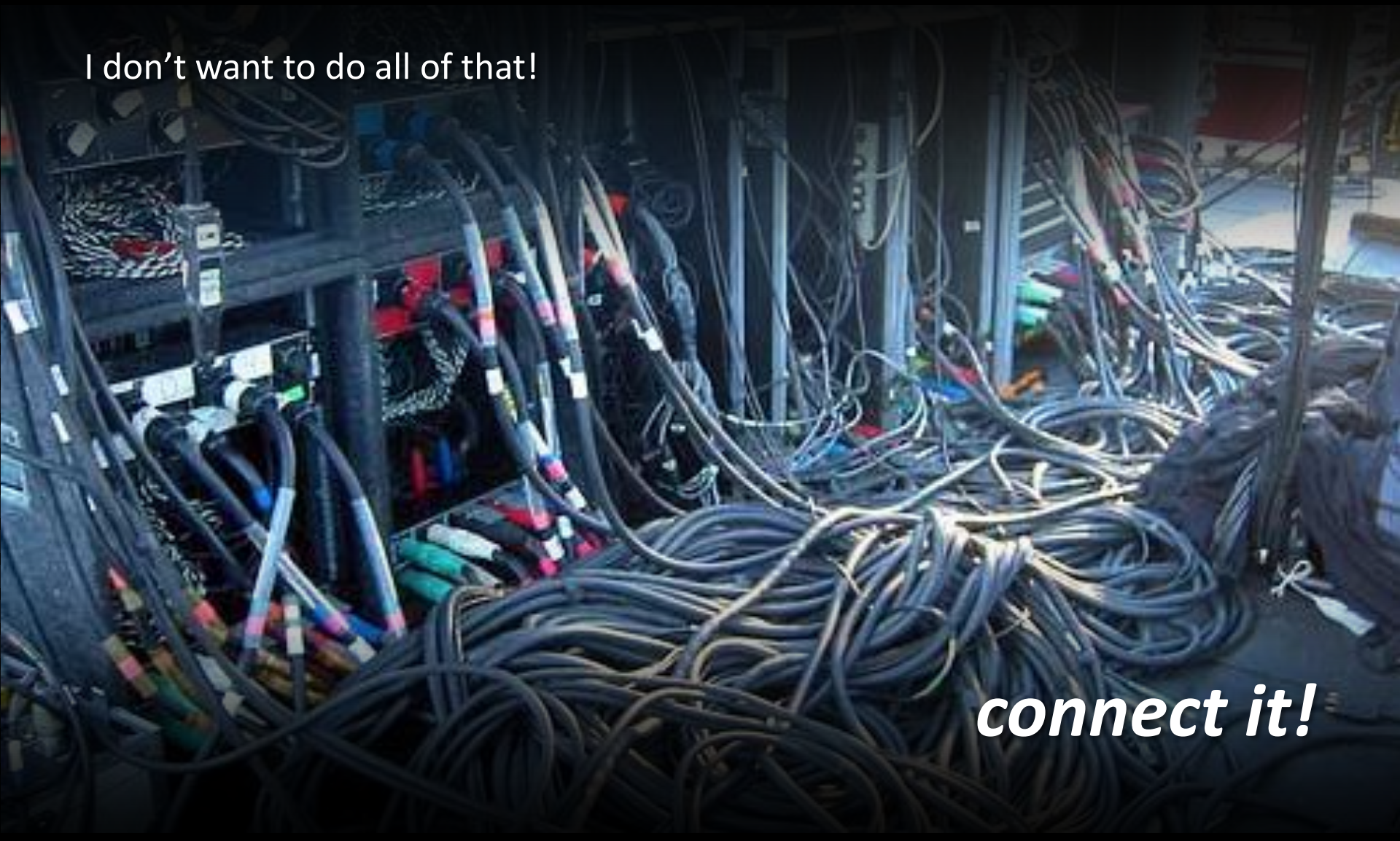


Image source: Webadicto!

Connecting the Engine

Finally, let's take a look on the interfaces an engine might have and find some good patterns on how to model them.

Read-only

Create a manager instance in the engine, which translates incoming data (Player Input, ..)

Write-only

Use a renderer to create a representation of required information (Graphics, Sound, ..)

Communication

No generic pattern. You will have to integrate it into the architecture (Physics, ..)

Connecting the Engine

Let’s take a look at the most popular (and most important) Engine interface tomorrow: The graphics!