Video game development Techniques and methods

adrien.allard.pro@gmail.com

Real time | Good practices | Data driven

Background

Game that places focus on military strategy at the level of movement and use of an entire nation state or empire's resources.

- Explore
- Expand
- Exploit
- Exterminate

Amplitude Studios

ES2
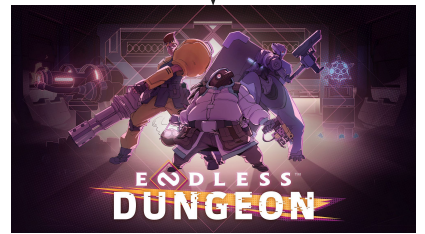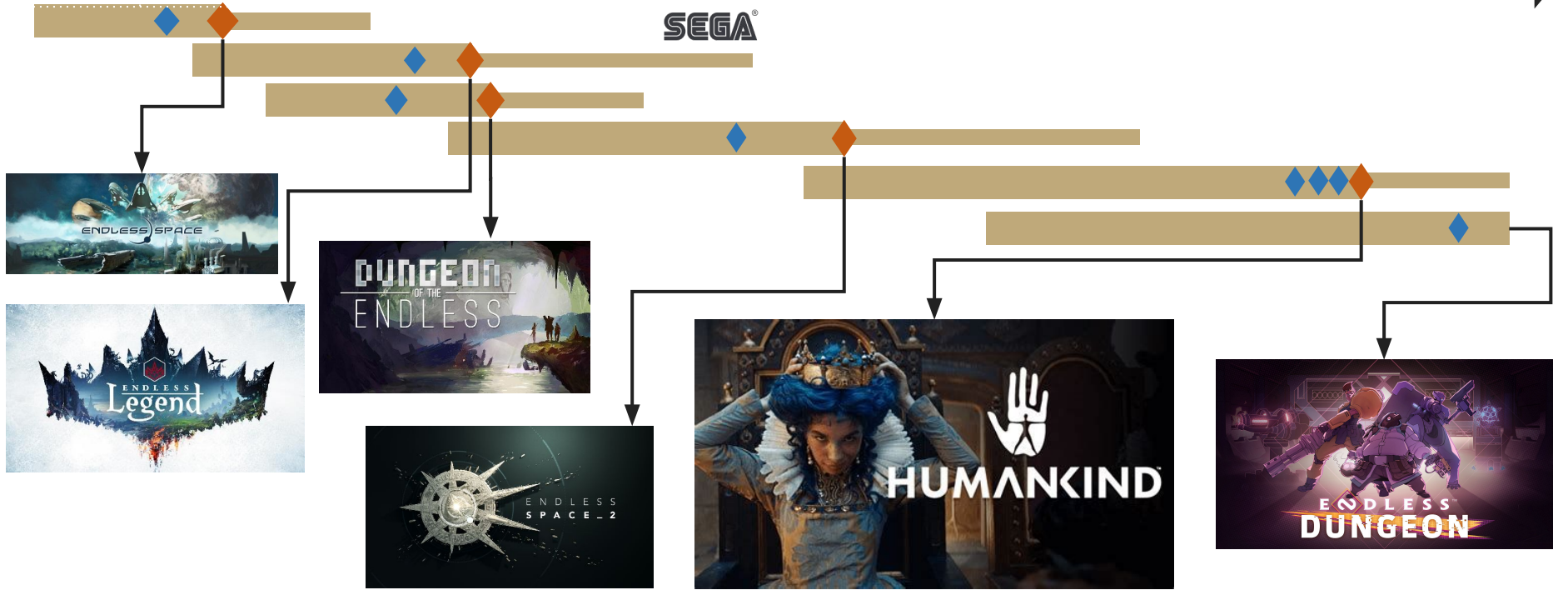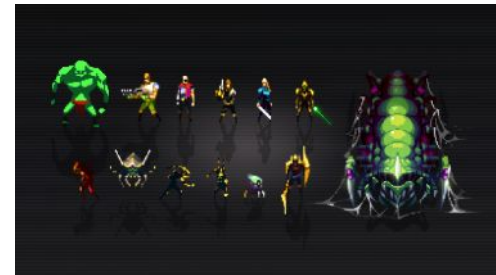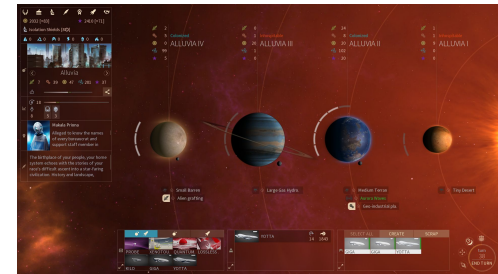
PATHFINDERS
LAUNCH TRAILER
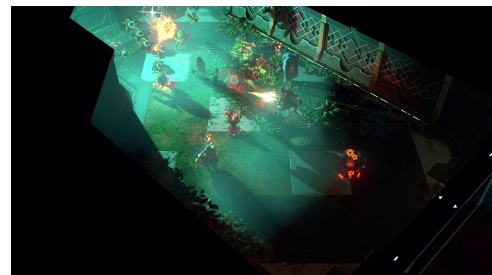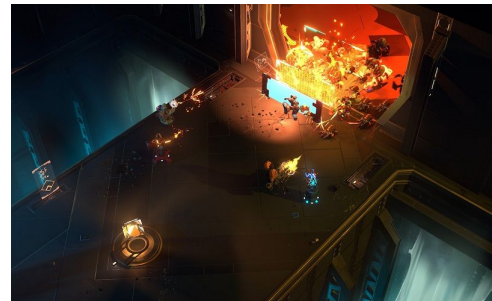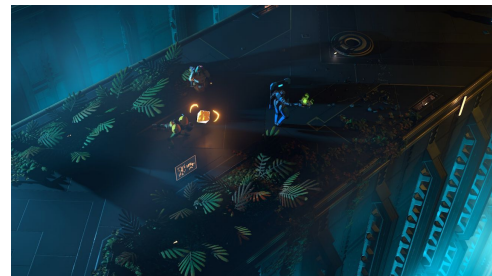
We use middlewares like Unity and Wwise.

We also develop our own tools and systems in C# and C++:
- Gameplay framework
- User Interface engine
- Custom render pipeline
- AI framework
- Native Animation system
- FX system

Real time

How to improve performances?

Real Time

Target: 60 frames per seconds (or 120 for VR)

- **16 milliseconds** to compute a frame (for both CPU and GPU).
- Rendering take at least 50% of the CPU time frame.
  - Less than 8 milliseconds to compute the rest of the game (Gameplay, AI, Network, Audio …).

When you have a performance issue, check which processor is slowing down your framerate to know where you need to make optimizations.

GPU side

- ~~Vertex count~~

- Fill rate

- Post processes (SSAO, FXAA, …)

- Custom Shaders

CPU side

- Algorithms with big complexity

- ~~Arithmetic operations~~

- Allocating memory

- Accessing hard drive data

- Bad uses of your game engine

- Draw calls

How to be more efficient?

- Instantiate the least frequently you can.

- Adapt your problem to your scope.

- Use the right data structure.

- Try to reduce the algorithm complexity.

- Pre-compute data and store it in memory …

- … but sometimes it's faster to recompute data.

- Change to a faster (even if it's less accurate) algorithm.

Heavy Algorithmic Problems

Try to reduce the algorithm complexity but be careful with your context.

- Algorithm 1: $O(n) = n^2$
- Algorithm 2: $O(n) = n$

Map height are in [40,150]

→ **Algorithm 1 is more efficient in our case !**

Duration by complexity

Take care of your constraints when you choose an algorithm.

Take care of your constraints when you choose an algorithm.

If you have a less than 100 elements, it's better to have a simple linear search than binary search.

Most of the time, KEEP IT SIMPLE !

Source: http://goo.gl/ub0CUg
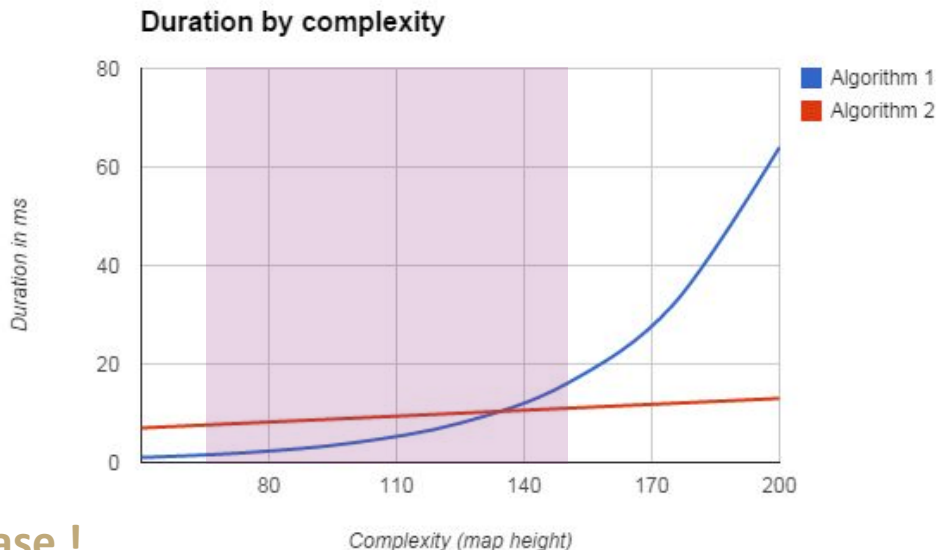
```
0100
0011
1001
```
Heavy Algorithmic Problems
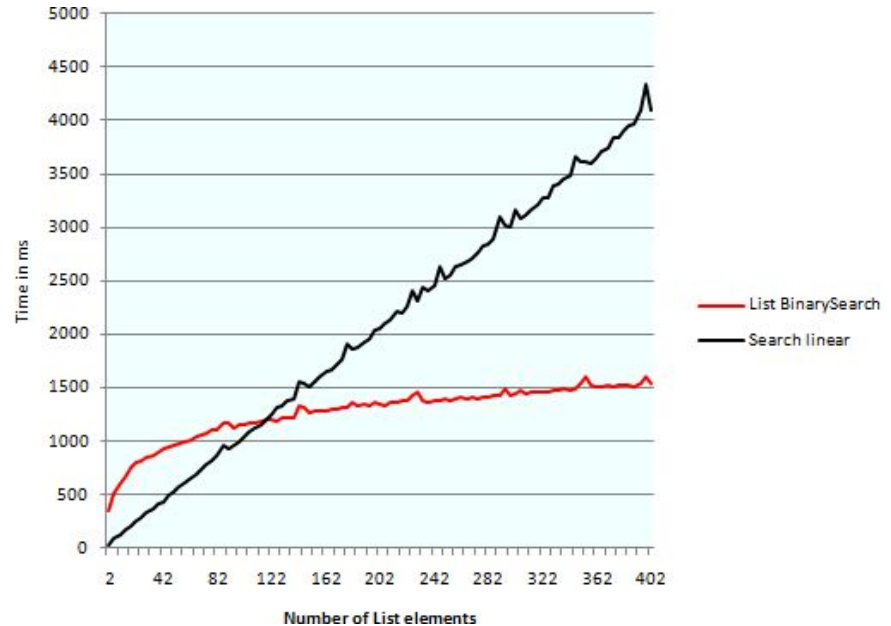
Retrieve objects (or components) can be very expensive !
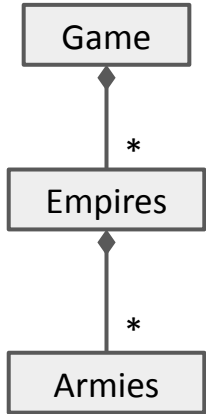
```
Army GetArmyAtPosition(int row, int column)
```

For 50 empires, 10 armies by empire it takes 500 iterations to retrieve this potential army.

```
Army GetArmyNearMe(int distance)
```

For a distance of 1 it takes 6 * 500 iterations.
For a distance of 2, 18 * 500 = 9000 iterations!

Game

*

Empires

*

Armies
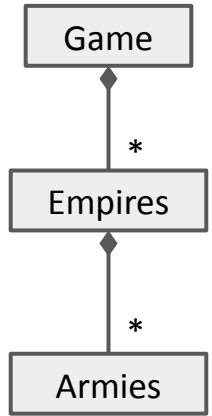
Solution 1

Instantiate a tile data structure with additional informations.

Problem
Aggregate data that are not related to each other.

```
World
Tile [,]
```

*

```
Tile
int row
int column
Army army
```
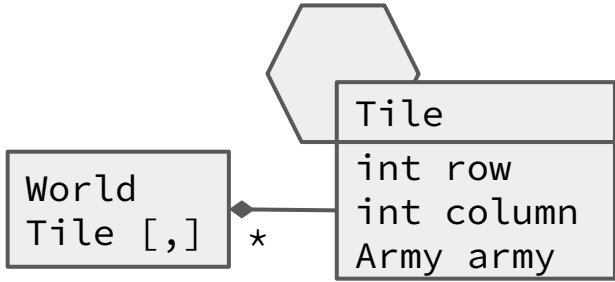
Solution 2

Instantiate a map data structure that contains the informations with a cheap access by position.

A map is a matrix of size rows*columns which contains the armies references indexed by position.

```
World
```

*

```
Maps
```

```
ArmyMap

Army[,] army
```
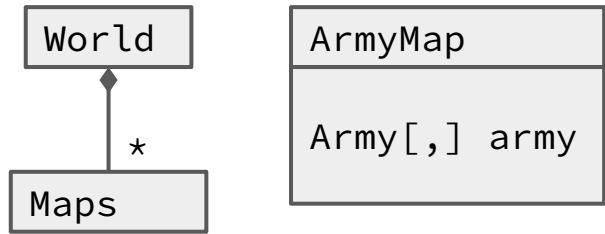
Data Structures

There is a lot of differents data structures that exists for a good reason. Find out what you need and what is the best solution for your problem.

Array	Cannot grow automatically. Very fast assignation speed.
List	Automatically growing array. A bit slower than array but still very fast.
Dictionary	Can retrieve value from key with O(1) to O(n) worst case complexity. Enumerate and insertion are expensive.
SortedList	Sorted generic list. Slowed on insertion.

and many others ...

Take care of your constraints when you choose a data-structure.

List	Dictionary
Retrieve element fast: O(n)	Retrieve element very fast: O(1) to O(n) in the worst case **(75 times faster)**
Insertion is fast. (graph)	Insertion is slow **(175% slower)**
Foreach is fast.	Foreach is slow **(57% slower)**

Unity Profiler

To figure out what takes time. Use the unity profiler!
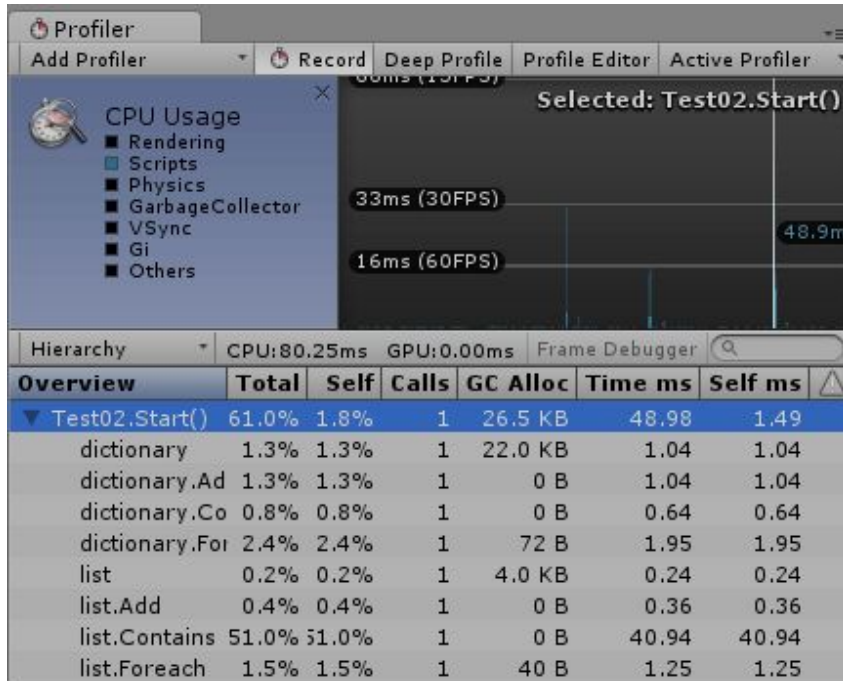

```
private List<int> list;

public void Start()
{
    Profiler.BeginSample("list");
    list = new List<int>(1000);
    Profiler.EndSample();
}
```

Unity Profiler

You can also use the Unity Profiler to see the GPU usage and the frame debugger to see how the frame is computed.

Your algorithms are not the only responsibles.

Be careful when you are using some features of your game engine.

Find out how things works !

The engine tools are designed to be used in a certain way.

Cube_002
493 verts, 456 tris   uv,uv2,colors

Cube_005
269 verts, 186 tris   uv,colors

Cube
24 verts, 12 tris   uv,uv2,colors
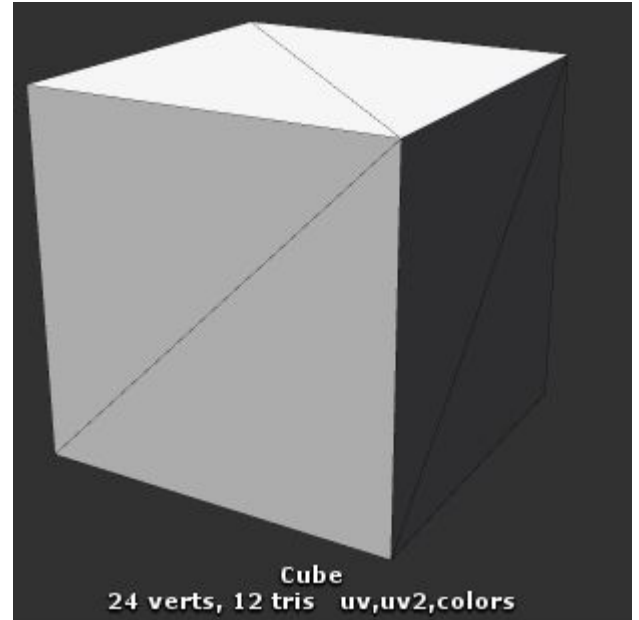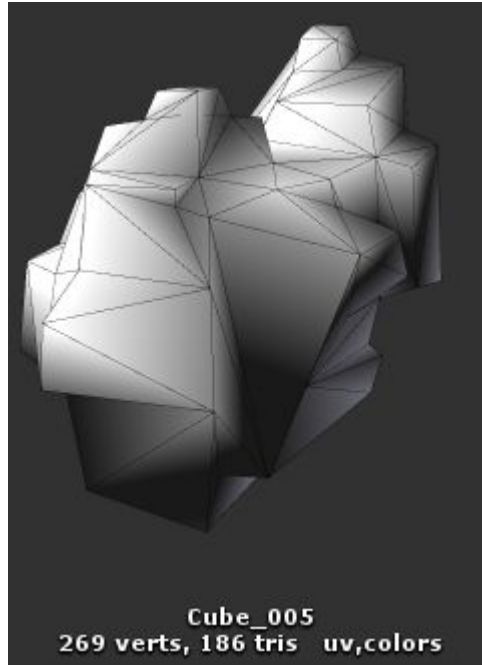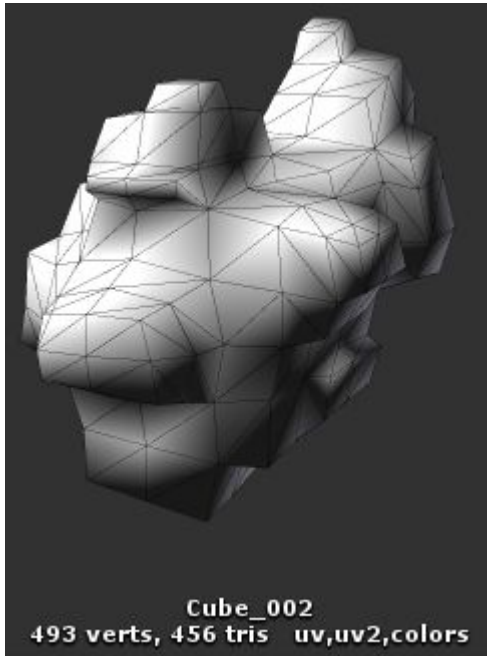
SendMessage and BroadcastMessage are slow. Don't use this functions!

```
public class ExampleClass : MonoBehaviour {
    private void ApplyDamage(float damage) {
        print(damage);
    }

    private void Example() {
        SendMessage("ApplyDamage", 42.0f);
    }
}
```

1000 calls	SendMessage	5.34ms	Direct calls	0.05ms
Tree with 4 levels and 4 children per level	BroadcastMessage	2.32ms	Direct calls	0.07ms

Definition: *An event in C# is a way for a class to provide notifications to clients of that class when some interesting thing happens to an object.*

Event is the c# implementation of the **observer design pattern**.
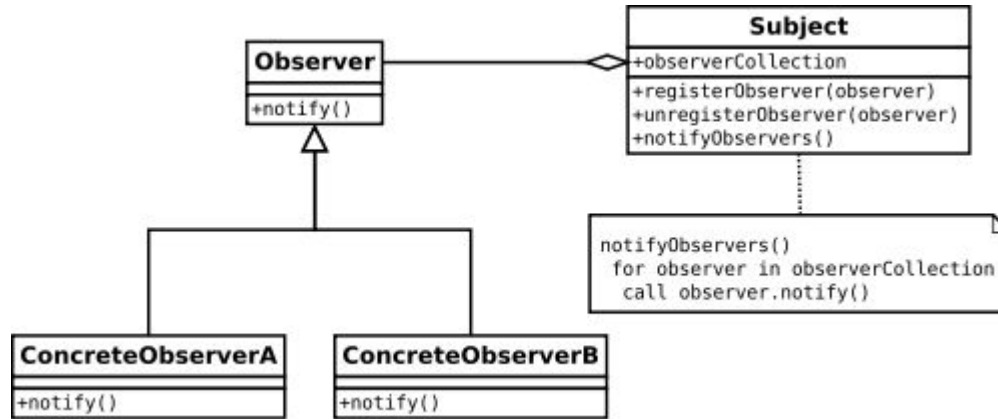
Declaration

```csharp
public class MyObservableObject
{
    public event System.EventHandler<System.EventArgs> OnSomethingAppend;

    public void SomethingAppend()
    {
        if (this.OnSomethingAppend != null)
        {
            this.OnSomethingAppend.Invoke(this, new System.EventArgs());
        }
    }
}
```

Register and unregister observer

```csharp
public class Observer
{
    public void RegisterObserver(MyObservableObject o)
    {
        o.OnSomethingAppend += this.SomethingAppendDelegate;
    }

    public void UnRegisterObserver(MyObservableObject o)
    {
        o.OnSomethingAppend -= this.SomethingAppendDelegate;
    }

    private void SomethingAppendDelegate(object sender, System.EventArgs args)
    {
        // Do stuff
    }
}
```

Definition

Polling, or polled operation, in computer science, refers to actively sampling the status of an external system as a synchronous activity.

Example

```
private void Update() {
    if (UnityEngine.Input.GetMouseButtonDown(0)) {
        // Do Something
    }
}
```

Static allocations

```
int n;
int x[10];
double m;
```

The compiler defines that the code requires 4 + 4 × 10 + 8 = 52 bytes on the **stack**.

This space is called stack because as functions get called, their memory gets added on top of existing memory.

As they terminate, they are removed in a LIFO (last in, first out) order.

Dynamic allocation

The function `void* malloc (unsigned int size)` requests `size` bytes of memory from the operating system, and returns the pointer to that location as a result.

This memory is assigned from the **heap** space.

This is called the heap space since it can be selected from any portion of the space that has not been allocated already.

While the stack remains nicely organized, memory in the heap tends to be more messy and all over the place. Hence the name.

Memory Allocation

Static allocation	Dynamic allocation
Size must be known at compile time	Size may be unknown at compile time
Performed at compile time	Performed at run time
Assigned to the stack	Assigned to the heap
First in last out	No particular order of assignment

Differences between statically and dynamically allocated memory

In C#

- Static allocation > Value type (int, float, struct, …)
- Dynamic allocation > Reference type (class)

A garbage collector takes care of the dynamic allocation.

Unlike in C++ class and struct are very different in c#.

Retrieving data from memory can take time.

1 cycle to read a register

4 cycles to reach to L1 cache

10 cycles to reach L2 cache

75 cycles to reach L3 cache

hundreds of cycles to reach main memory.

Between **0.08 and 0.16 ms** to reach SSD.

Between **0.2 and 0.8 ms** to reach HDD.

In Unity:

- **0.04 ms** to compute the arithmetic mean of 15 000 elements.
- **1 ms** to instantiate 15 000 empty classes.
- **3 ms** to instantiate 15 000 classes that contains a float value.
- **1 ms** to instantiate 100 empty prefabs.
- **2.8 ms** to instantiate 100 cube prefabs.

Test processor : Intel core i5-3570K @ 3.4 Ghz

Shoot'em up Project

In the shoot them up project you need to instantiate some objects during game:

- A bullet each time an avatar is firing
- An enemy each time a new one is spawning

If a lot of instantiations are processed at the same time, it could produce some freezing of your game (specially if you run on small device, like smartphone).

You need to handle the worst case ! (spawning of X avatar and Y bullets at the same frame).

You must control the instantiation of your objects.

```csharp
public interface IPeople {

    string GetName();

}


public class Villagers : IPeople {

    public string GetName() { return "Village Guy"; }

}


public class CityPeople : IPeople {

    public string GetName() { return "City Guy"; }

}


public enum PeopleType {

    RURAL,

    URBAN

}
```

```
public class Factory {

    public IPeople GetPeople(PeopleType type) {

        IPeople people = null;

        switch (type)

        {

            case PeopleType.RURAL :

                people = new Villagers();

                break;

            case PeopleType.URBAN:

                people = new CityPeople();

                break;

            default:

                break;

        }

        return people;

    }

}
```

The factory method pattern is a creational pattern which uses factory methods to deal with the problem of creating objects without specifying the exact class of object that will be created.
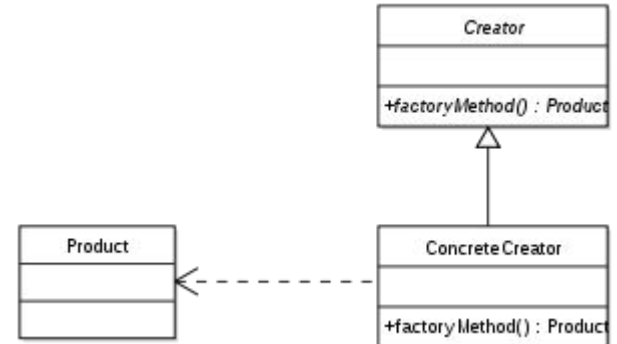
Recycling

When a bullet hit something, don't destroy it. Just disable it and keep it ready to be reused.

This recycling logic can be put in the factory. So you just have to ask for an instance of bullet, the factory will recycle or create a new bullet for you.

Optimization: You can make an estimation of the number of bullets needed in your game and then pre instantiate it while the game is loading.

You can do the same thing for avatars.
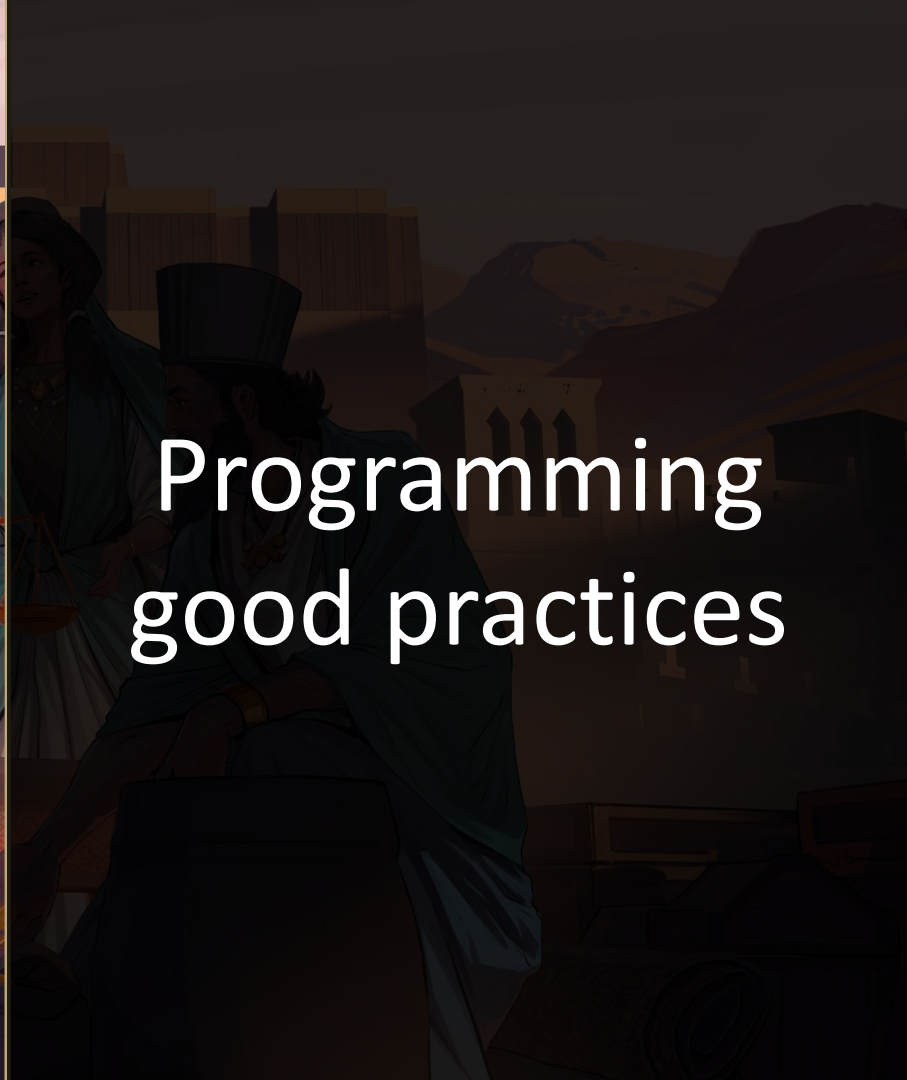
Programming good practices

Defensive Programming

Defensive programming is an approach to improve:

- General quality - reducing the number of software bugs and problems.

- Making the source code comprehensible - the source code should be readable and understandable so it is approved in a code audit.

- Making the software behave in a predictable manner despite unexpected inputs or user actions.

You can't prevent everything ! and if you think you can, you can't prevent the future!

You must protect your code from errors.

Defensive Programming

- **Intelligent source code reuse**
  If existing code is tested and known to work, reusing it may reduce the chance of bugs being introduced.

- **Secure input and output handling**
  "never trust the client"

- **Low tolerance against "potential" bugs**
  "I'm not aware of all problems. I must protect against those I do know of and then I must be proactive!"

Defensive Programming

Assertions

```
public void Assert(bool condition, string message) {
    if (!condition) {
        UnityEngine.Debug.LogError(message);
    }
}
```

Check your results to test if their values are correct:

```
public float Abs(float number) {
    float result = number < 0 : -number : number;
    UnityEngine.Debug.Assert(result >= 0, "result must be positive");
    return result;
}
```

Defensive Programming

Exceptions

Check your inputs to test if their values are handled by your code:

```
public float Abs(float number) {
    if (float.IsNaN(number)) {
        throw new InvalidArgumentException("number", "The number must have a valid
value");
    }

    float result = number < 0 : -number : number;
    System.Collection.Generic.Assert(result >= 0);
    return result;
}
```

Exceptions

Then, you can catch exceptions when you use your function:

```csharp
public float Abs(float number1, float number2) {
    float result = 0f;
    try {
        result = Abs(number1) + Abs(number2);
    } catch (System.Exception exception) {
        // Handle your error here
        throw; // If your want to forward the exception.
    }
    return result;
}
```

Never assume "It cannot happen"

Handle all cases:

```
public void DoSomethingWithStuff() {
    object stuff = CreateStuff();
    if (stuff == null) {
        Log("Error while creating stuff instance");
        ManageErrorToAvoidCrash();
        return;
    }

    stuff.DoSomething();
}
```

Code readability

Coding style

It is important to have a common coding style in a project, it makes the code a lot more understandable. Use tools to help you (like StyleCop).

Naming

Be careful when you name your class, methods and variables.

If theirs name are carefully choiced, you'll don't need to read the code to understand what it does.

```
float energyLostPerFiring = 10f     // Better than "float loss = 10f";
```

Documentation

Write documentation, specially when your code is complicated.

Code review is systematic examination of computer source code.

It is intended to find mistakes overlooked in software development, improving the overall quality of software.

Some web-based source control services like GitHub have integrated a lot of tools to handle code reviews.

Keep in mind that your code will be read by other people.

It's dangerous to start writing complicated code without using solid reflection tools.

Expose your ideas to other people
Confront your ideas to another brain is a strong way to ensure that your are on the right way (even if the other people don't know what you are talking about -> canard en plastique).
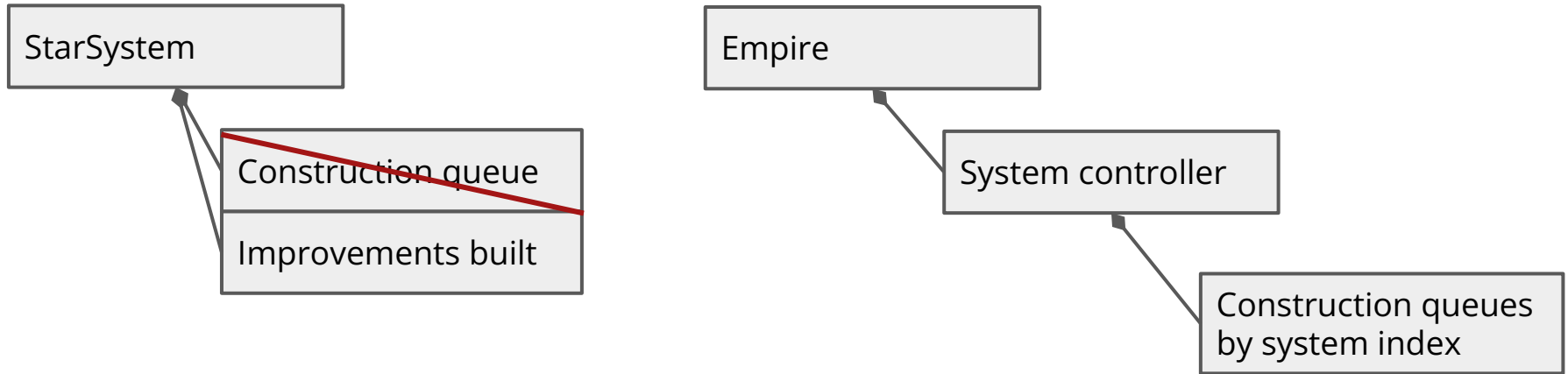
Pen & paper or blackboard
Make schema and write your ideas, you may discover that it is stupid before beginning to implement it.

UML
Very useful tool to talk about architecture with other programmers (because it's a common description language). You can try StarUML.
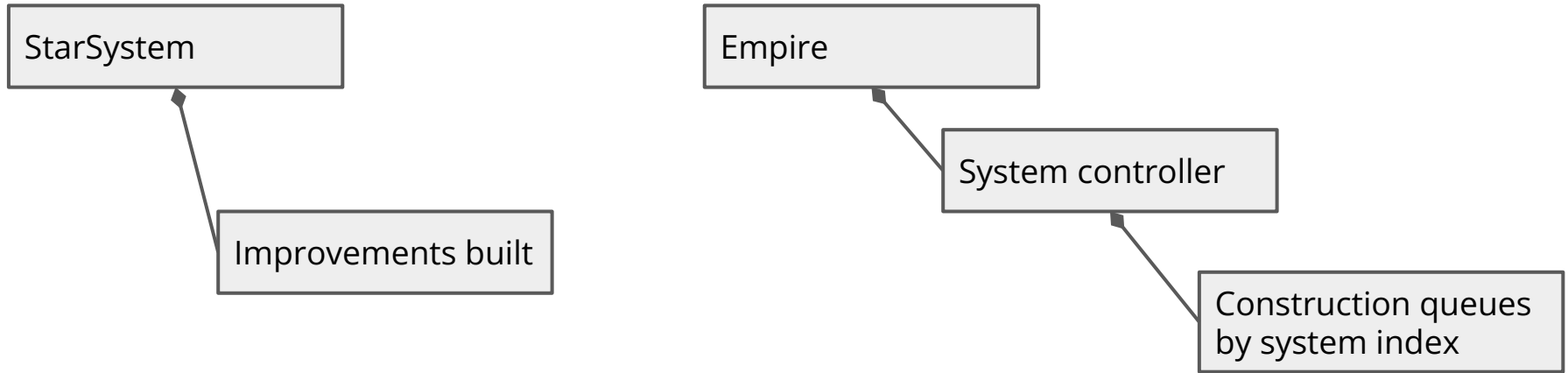
Endless Space construction queue

- Define who is really in charge of a specific functionality.
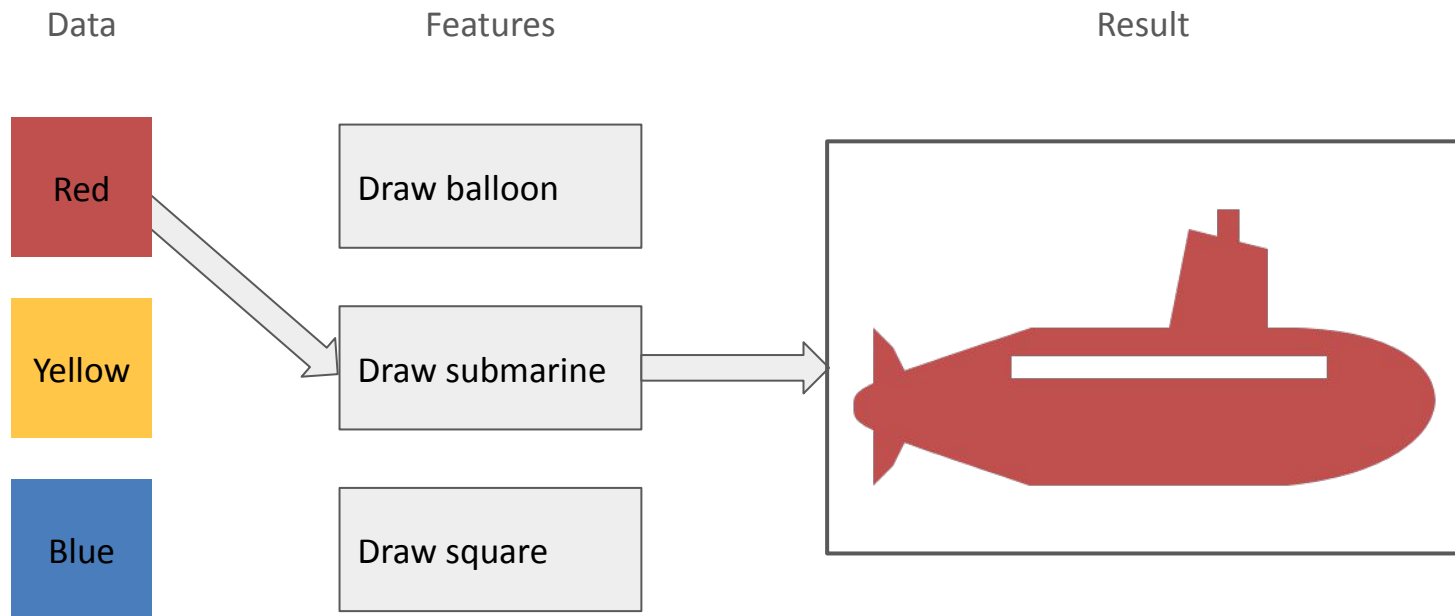- Split your functionalities (and remove dependencies if possible).

Data Driven

A data-driven game is a game that exposes a large amount of functionality outside of code and lets the data determine the behavior of the game.

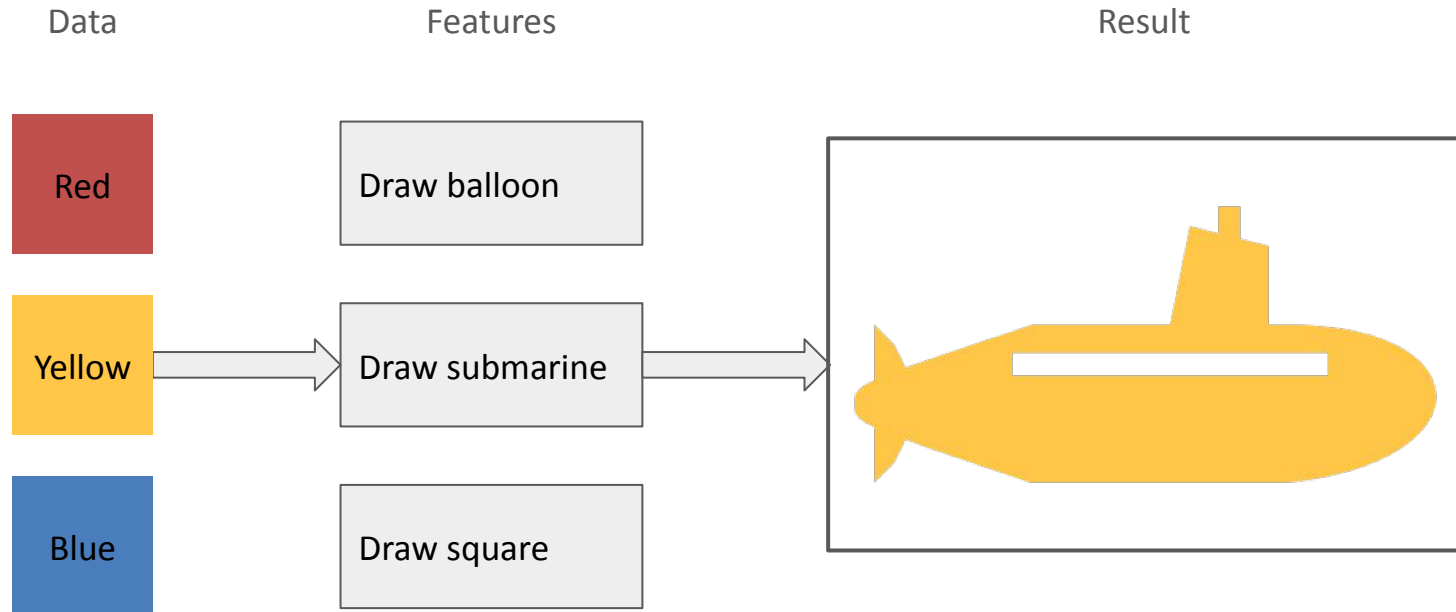| Data	Features	Result

A data-driven game is a game that exposes a large amount of functionality outside of code and lets the data determine the behavior of the game.

Data	Features	Result

Red	Draw balloon	
Yellow	Draw submarine	
Blue	Draw square	

A data-driven game is a game that exposes a large amount of functionality outside of code and lets the data determine the behavior of the game.

Pro

+ Re-Use

+ Job Specialization

Cons

- Data management layer cost

- Abstraction cost

Humankind narrative event system

- Title
- Image
- Narrative text
- Prerequisites
- Choices
  - Title
  - Narrative text
  - Effects
  - AI info

Code Approach

- Require a programmer to add content
- Not easy to made this system ready for modding

```csharp
abstract class NarrativeEvent
{
    private List<Choice> choices = new List<Choice>();

    abstract void RegisterChoices(Game game);

    abstract bool CheckPrerequisites(Game game);
}

class SomeEvent : NarrativeEvent
{
    public override RegisterChoices(Game game)
    {
        this.choices.Add(new Choice
        {
            Title = "Cats",
        })

        this.choices.Add(new Choice
        {
            Title = "Furet",
        })
    }
}
```
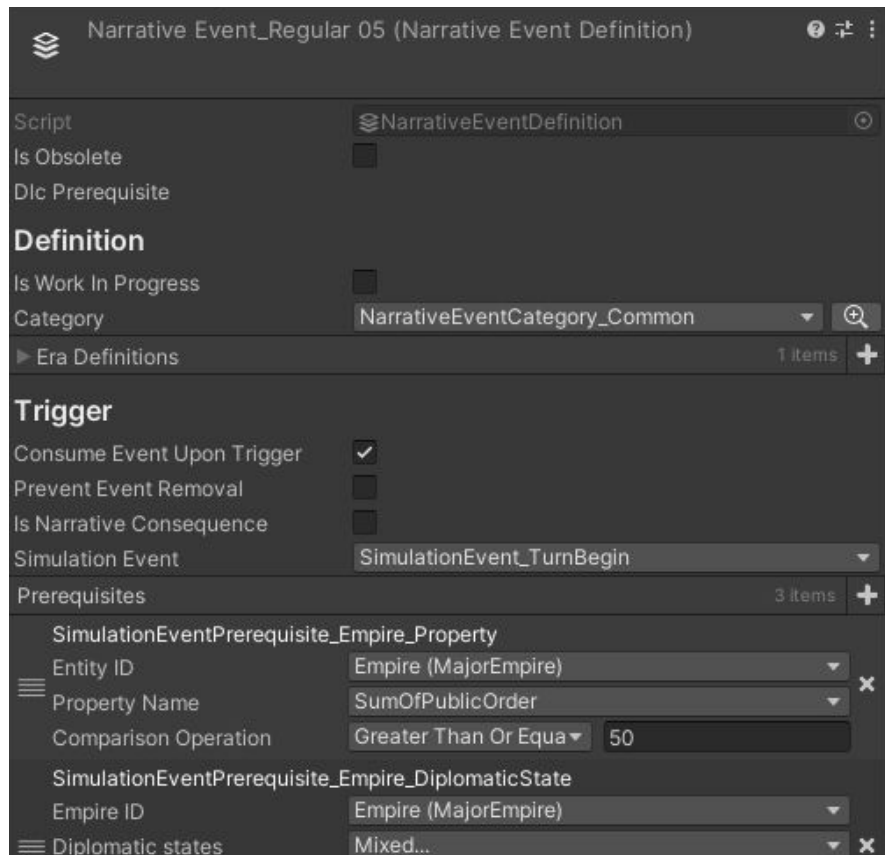
Data Driven Approach

- Require a programmer only when you need to add new prerequisites/effect type
- You can give the same tool to the player for modding
- Give the power to narrative designer to add new content in the game

```
class NarrativeEventManager
{
    // Code that interpret NarrativeEvent data
}
```

NarrativeEvent
Database

Data format - Scriptable Object


```csharp
using UnityEngine;

[CreateAssetMenu(fileName = "Data", menuName =
"ScriptableObjects/SpawnManagerScriptableObject", order = 1)]
public class SpawnManagerScriptableObject : ScriptableObject
{
    public string prefabName;

    public int numberOfPrefabsToCreate;
    public Vector3[] spawnPoints;
}
```

```xml
<NarrativeEvent Name="NarrativeEvent_Regular05">
    <Title>L'accapareur de champignons</Title>
    <Prerequisites>Food > 10</Prerequisites>
    <Choices>
        <Choice Title="Ban">
            <Title>Bannir</Title>
            <Effect>-25% DefenseResearchCost</Effect>
        </Choice>
        <Choice Title="Protect">
            <Title>Protéger</Title>
            <Effect>Abundance for 10 turns</Effect>
        </Choice>
        <Choice Title="Avoid">
            <Title>Garder à distance</Title>
            <Effect>+2 science</Effect>
        </Choice>
    </Choices>
</NarrativeEvent>
```

Shoot'em up Project

Data driven level design

We want to describe our level in XML files.

The game will then read these files to create the real levels and run the game.
A level design can contains many informations:
- **Enemies**
    - Type
    - Position
    - Spawn time
- **Obstacles**
- ...

Serialization is the process of converting an object into a form that can be readily transported.

The central class in XML serialization is the `System.Serialization.Xml.XmlSerializer` class, and the most important methods in this class are the `Serialize` and `Deserialize` methods.

XmlSerializer serializes only the public fields and property values of an object into an XML stream, use it when the XML stream is expected to conform to a known XML Schema (else you can use XmlWriter and XmlReader to manually serialize/deserialize your data).

Sources: https://goo.gl/j85m99

C# class

```csharp
public class OrderForm
{
    public DateTime OrderDate;
}
```

XML file

```xml
<OrderForm>
    <OrderDate>12/12/01</OrderDate>
</OrderForm>
```

With XmlSerializer you can:

- Serialize classes that implement ICollection or IEnumerable.
- Generate a XSD schema files from your code, then check if your data files are well formed.
- Specify whether a field or property should be encoded as an attribute or an element.
- Specify the name of an element or attribute if a field or property name is inappropriate.

⚠️ A class must have a default constructor to be serialized by XmlSerializer.

```
[XmlRoot("Order")]
public class OrderForm
{
    [XmlAttribute("Date")]
    public DateTime OrderDate;

    [XmlElement("Value")]
    public float FloatValue;
}
```

XML file

```
<Order Date="25/09/2015">
    <Value>42</Value>
</Order>
```

Code features needed:

- Ability to spawn an enemy with specific attributes.

- Create a "Game Manager" to manage level and progression.

- Meta-game system that check if the current level is finished and
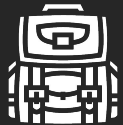 launch the next one (keeping score and other persistent infos).

Then …

… Just add some XML files to add more

content to your game !

Conclusion

Takeaway

- Keep your code as simple as possible.

- Learn how to use your game engine.

- Beware of performances. Do not assume! Try and measure!

- It's always CPU vs Memory problem.

- Data driven is good!

- Keep your code as simple as possible. (It's very important!)

- Practice programming!

THANK YOU!

Questions?

C#

http://www.codeproject.com/Articles/375166/Functional-programming-in-Csharp
http://www.codeproject.com/Articles/286255/Using-LINQ-Queries
https://docs.microsoft.com/fr-fr/dotnet/csharp/fundamentals/object-oriented
http://www.dotnetperls.com

Algorithmics

http://www.redblobgames.com/

Data-Driven design

http://gamearchitect.net/Articles/DataDrivenDesign.html
http://www.cs.cornell.edu/database-OLD/games/

Annex - Documentation

C# Memory management for Unity

http://www.gamasutra.com/blogs/WendelinReich/20131109/203841/C_Memory_Management_for_Unity_Developers_part_1_of_3.php
http://www.gamasutra.com/blogs/WendelinReich/20131119/203842/C_Memory_Management_for_Unity_Developers_part_2_of_3.php
http://www.gamasutra.com/blogs/WendelinReich/20131127/203843/C_Memory_Management_for_Unity_Developers_part_3_of_3.php

Memory hierarchy

http://en.wikipedia.org/wiki/Memory_hierarchy
http://www.sisoftware.co.uk/?d=qa&f=mem_hsw
http://www.gdcvault.com/play/1014645/-SPONSORED-Hotspots-FLOPS-and

Shoot'em up game with Unity:

https://github.com/Tichau/Schmup

Tetris game (standalone/web/android) with Unity:

https://github.com/Tichau/Tetris

Adrien Allard

Project Technical Director

adrien.allard.pro@gmail.com

Humankind   **>** **https://tinyurl.com/humankindgame**