



ATELIER

Créer une interface graphique en pur code

* Introduction *

Type d'atelier :

Centre Technologique

Technos :

Lua / Love2D

C# / Monogame

Lien de l'atelier :

<https://www.gamecodeur.fr/liste-ateliers/atelier-gui-pur-code/>

Sommaire

Introduction et objectifs	3
Vocabulaire	3
Approche théorique	3
Elements d'interface courants	3
Coder ou utiliser une lib	4
Conception : ma proposition	5
Dessiner les éléments	6
Réagir aux interactions	7
Pour aller plus loin	8
Conclusion sur la partie théorique	9

Introduction et objectifs

Dans cet atelier vous allez apprendre :

- Ce qu'est une Interface Utilisateur orientée souris
- Comment architecturer votre code pour créer un système d'interface évolutif
- Créer des panels
- Créer des labels textes
- Créer des boutons et gérer les événements associés
- Créer des cases à cocher et tester leur état
- Créer des groupes, et gérer leur visibilité

Vocabulaire

Voici les termes que vous rencontrerez sur Internet et dans le milieu professionnel pour désigner l'interface graphique :

UI / User Interface

GUI / Graphical User Interface

HUD / Head Up Display

UX / User Experience

Il y a peu de différence entre ces termes, même si HUD est plus utilisée pour les interface "in game" (pendant le gameplay). Mais globalement ce ne sont que des querelles de clochers.

Techniquement, notamment pour nommer les bibliothèques / frameworks qui fournissent des composants d'interface, on utilise le plus souvent "GUI".

Approche théorique

Je vous propose une approche théorique pour commencer. Vous pouvez même approcher cet atelier comme un projet d'exercice et programmer sans consulter les sections "mise en pratique".

Dans cette partie je vais vous former aux bases fondamentales d'une GUI et vous guider dans l'approche de la programmation de cette GUI.

Elements d'interface courants

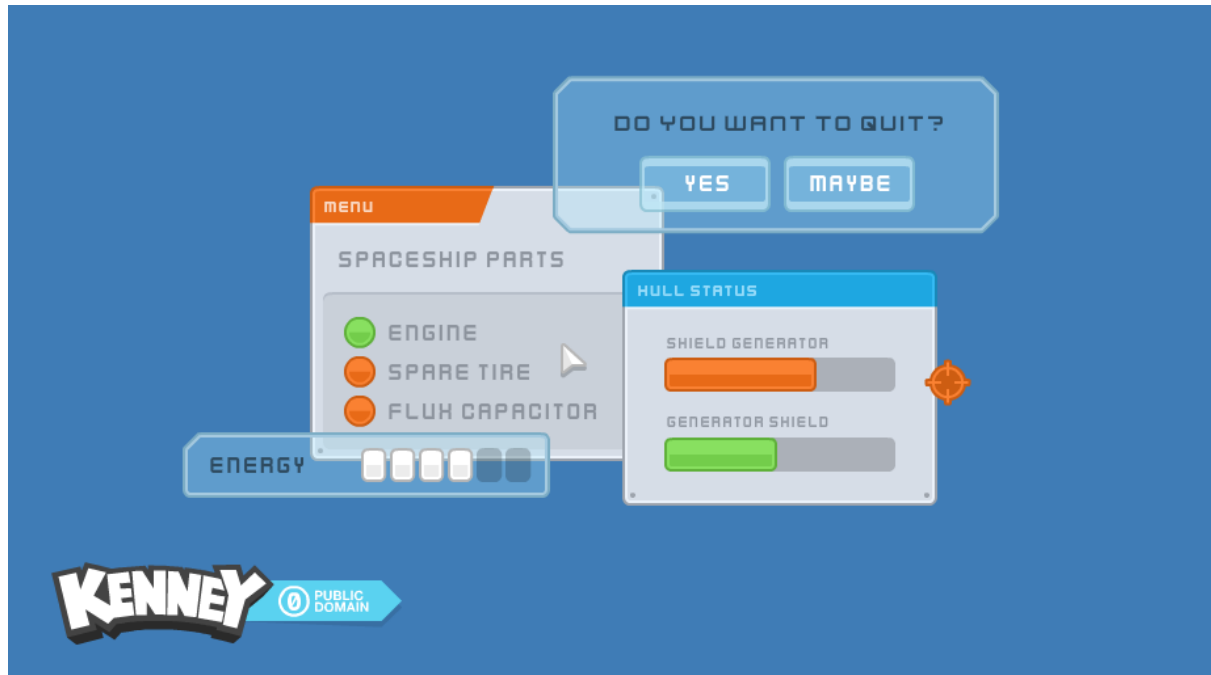
Pour un jeu vidéo traditionnel, type arcade, nous avons essentiellement besoin de :

- panneaux (ils servent de cadres)
- labels (texte simple, pas d'interactivité)
- boutons
- cases à cocher

Pour des interfaces plus avancées :

- barres de progression (progress bars)
- listes (déroulées ou déroulantes)
- sliders
- et autres éléments ésotériques, notamment sur mobile...

Nous traiterons dans cet atelier des labels / boutons / cases à cocher ainsi que les barres de progression.



Cela sera une bonne base technique pour vous permettre de développer, si besoin, des éléments plus complexes.

Coder ou utiliser une lib

Pour coder un système d'interface graphique, vous avez 2 choix :

- utiliser une librairie existante (elle sera bien souvent "oversized" par rapport à vos besoins et vous devrez apprendre à vous en servir).
- coder votre propre système

Bien entendu, c'est la 2ème option qui est traitée dans cet atelier.

Pour moi, c'est important de savoir coder un système d'interface, par culture personnelle et par besoin !

On a souvent besoin de bien peu, et c'est codable facilement. De plus, il y a des chances que votre interface soit plus adaptée, plus personnelle, si vous codez votre propre système.

Sans parler des performances.

Néanmoins, si vous souhaitez partir sur quelque chose d'existant, bonne chance.


Si vous codez avec une lib ou un framework (Love2D, Monogame, etc.), le choix sera difficile car les projets sont souvent peu maintenus. Choisissez la lib la plus légère possible, et sans dépendances externes. Et bien sûr, quelque chose dont le code est disponible et libre de droits.


Avec Unity le souci ne se pose pas car le système de GUI est intégrée au moteur. Ce n'est pas sans défaut : les performances ne sont pas toujours au rendez-vous.

Conception : ma proposition

En concevant votre système d'interface, voici les choses que je vous conseille de prendre en compte :

- Organisez les éléments par groupe, cela permet par exemple d'afficher / cacher des sous interface, des menus, etc.
- Pour les événements, travaillez avec des références de fonctions (c'est le bouton qui appellera votre fonction par exemple). C'est plus simple.
- Si vous codes en POO, faites les choses proprement afin que le système soit évolutif (on va voir ça ensemble plus loin)

	<p>En Lua les choses seront relativement simples : on travaillera avec des listes imbriquées. Exemple :</p> <ul style="list-style-type: none">- GUI -> Groupes -> Elements
---	---

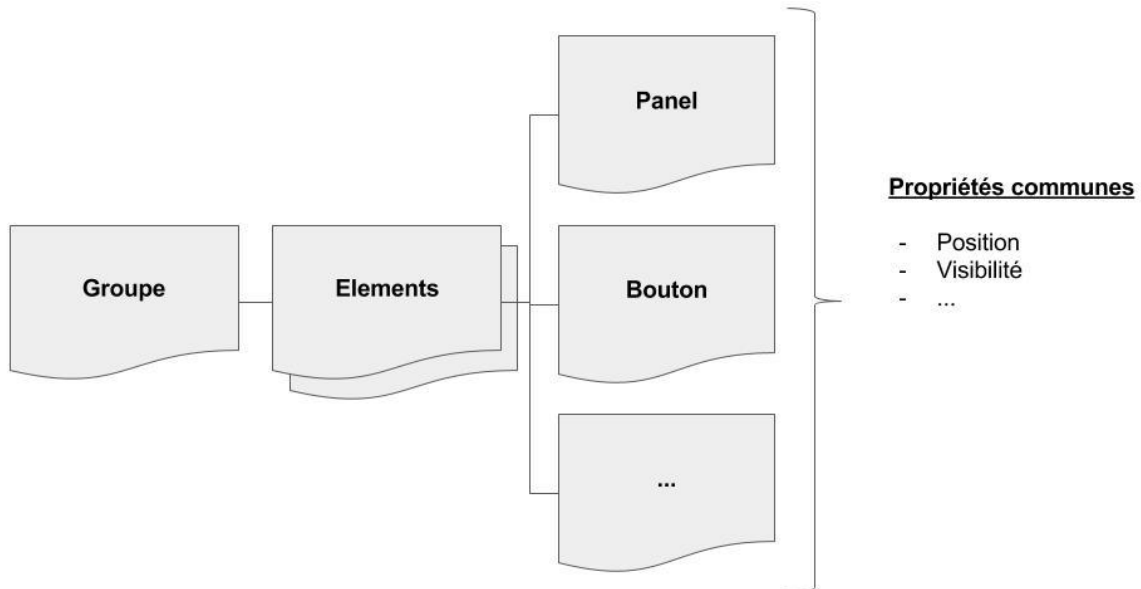
	<p>En C# on va partir sur une collection d'instances, basées sur une classe de base et proposant des méthodes virtuelles pour les comportements spécifiques.</p>
---	---

Voici ma proposition :

Nous manipulons des groupes et des éléments.

- Un élément représente un type abstrait d'élément de GUI
 - En Lua, ce sera la table de base
 - En POO ce sera la classe de base
- Un élément possède des propriétés / fonctions communes à tous les éléments
 - Position (x et y)
 - Visibilité (booléen)
 - Fonction d'affichage, de mise à jour, etc.
 - Eventuellement des événements communs (voir plus bas)

- Un groupe contient des éléments
- Le groupe propose les fonctions :
 - Ajouter un élément
 - Changer la visibilité du groupe
 - Dessiner le groupe



Nous dérivons ensuite de la base (élément) pour nous spécialiser :

- Panel : qui aura un type, une image, ...
- Texte : qui aura une chaîne de caractère, une police de caractère, ...
- Bouton : qui aura un type, des images, un texte, et des événements spécifiques
- Case à cocher : qui aura des images, un état, et des événements spécifiques
- Barre de progression : qui aura des images, une valeur max, et une valeur courante

Pour programmer ce genre de choses, je travaille par raffinement successif. Par exemple au départ mes éléments affichent une simple boîte et gèrent un événement basique, je teste que ça marche, puis j'ajoute les images, je crée les éléments plus avancés, etc.

Dessiner les éléments

Pour dessiner des éléments de UI, plusieurs choix s'offrent à vous :


- Utiliser des éléments graphiques (images)
- Utiliser des primitives (lignes, rectangles, textes)
- Gérer les deux (avec une propriété "type" pour les distinguer)



Pour Lua :

Nous utiliserons Love2D pour la partie graphique.

	Love2D couvre à la fois notre besoin pour l'affichages d'images et de primitives.
--	--

	<p>Pour C# :</p> <p>Nous utiliserons Monogame pour la partie graphique.</p> <p>Monogame ne propose pas de primitives, nous allons donc devoir coder le minimum nécessaire.</p>
---	---

Notes techniques pour l'affichage du texte :

Pour afficher le texte, notamment des boutons, regardez ce que propose votre framework. La majorité proposent l'affichage de texte avec une police de caractère Truetype. Attention cependant aux performances. Si la police est traitée "en direct" pendant l'exécution, cela peut être gourmand en CPU. Monogame, par exemple, convertie la police en bitmap ce qui vous facilitera la vie. L'inconvénient c'est qu'il faut plusieurs versions de la police pour gérer plusieurs tailles.

A noter également que selon les langues que vous voulez proposer, attention au choix de la police de caractère !

Vous proposez le français ou autres langues latines : vérifiez qu'elle gère les accents.

Vous voulez proposer le russe : vérifiez qu'elle gère les caractères cyrilliques !

Vous voulez proposer le danois : vérifiez qu'elle gère les caractères æ, ø et å !

Vous voulez proposer le japonais, et autres polices non alphabétiques : préparez-vous à souffrir !

Cet atelier ne couvrira pas les polices spéciales (japonais, chinois, etc.), rapprochez-vous des communautés du framework que vous utiliserez pour avoir des pistes.

Conseil de la mort :

N'intégrez pas le texte sous forme d'image, même si votre infographiste trouve ça "plus cool" parce qu'il peut ajouter des effets au texte. Pourquoi ? Parce que vous ne pourrez pas traduire votre jeu à moins de refaire toutes les images ! Utilisez plutôt du texte pur, avec une police Truetype (tous les frameworks proposent des fonctions pour afficher du texte). C'est moins fun, mais c'est comme ça. Voir plus loin dans cet atelier pour des exemples.

Réagir aux interactions

Pour réagir aux interactions, nous avons besoin de lister les événements qui peuvent se produire avec nos éléments de GUI.

Ensuite, pour réagir à ces événements, nous utiliserons les appels de fonctions (pointeurs de fonctions ou delegates) et nous assurerons que ces événements sont appelés correctement (une seule fois par clic, etc.).

Voici les événements courants que je vous propose de prendre en compte :

Événement	S'applique à...
Hover (survol de la souris) : déclenché quand on rentre et quand on quitte l'élément. Utile pour afficher une aide par exemple.	Panneaux
Clic (clic à la souris) : déclenché quand l'utilisateur clique.	Bouton Case à cocher
Changement d'état : déclenché quand la valeur affichée par l'élément change.	Case à cocher Barre de progression Texte

Note concernant le curseur souris :

Pour la gestion de la souris, la plupart des Frameworks n'affichent pas le curseur par défaut. Il faut donc au préalable trouver la fonction pour rendre le pointeur visible.

Il faudra également gérer précisément l'état de la souris, par exemple pour ne déclencher qu'une seule fois l'événement "Click", on devra ainsi attendre que le bouton de la souris soit relâché pour rendre possible la prise en compte d'un prochain événement Click.

Le principe théorique est :

- Test si le bouton est enfoncé
- Si il n'était pas enfoncé avant : événement
- Mémorise l'état du bouton pour la prochain itération

Je vous déconseille fortement de remplacer le pointeur souris par une image (du genre, vous n'affichez pas le curseur système et à chaque draw, vous dessinez une image à la position de la souris). C'est beau, mais ça peut poser des soucis de lag ou une grosse sensation d'inertie sur les configurations lentes.

Si vous prévoyez tout de même un curseur graphique, ajoutez une option pour le désactiver facilement.

Pour aller plus loin

On pourrait hiérarchiser les éléments entre eux. Un bouton pourrait par exemple être le "fils" d'un panel, etc. Le déplacement d'un élément parent pourrait alors impacter la position de

ses enfants. On parle de "position relative". Je ne couvrirai pas cette approche dans cet atelier mais je vous invite à le faire si vous voulez vous challenger.

Vous pouvez aussi prévoir une version "clavier" de votre interface :

- en associant des raccourcis clavier à chaque bouton, et en affichant le raccourci dans le coin du bouton par exemple
- en gérant un système de focus (comme dans une interface type Windows) mais là c'est vraiment aller loin et ce n'est ni simple pour vous, ni forcément ergonomique.

Conclusion sur la partie théorique

Je crois avoir abordé le principal. Bien entendu le sujet est sans fin, mais avec tout ça vous avez des bases. On va voir maintenant la mise en pratique.

Rappel : vous pouvez ne pas consulter la mise en pratique et tenter par vous même de coder une GUI sur la base de l'enseignement théorique que je vous ai proposé.

Je vous conseille d'enchaîner sur la mise en pratique Lua qui contient encore plus de détails (même si vous ne codez pas en Lua).

A suivre : Mise en pratique (voir la page de l'atelier)