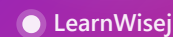


Canvas interactif : état, saisie, hit testing et performances

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Transformer un Canvas en éditeur : la boucle Modèle, Saisie, État, Rendu, les événements pointeur et tactiles issus de Control, la conversion écran vers monde pour que le hit testing survive au panoramique et au zoom, un état de scène par session plutôt que des champs statiques, l'élimination hors champ, et le moment où une boucle côté client remplace les allers-retours serveur. Ce que cela signifie pour un développeur .NET expérimenté qui construit déjà des écrans Wisej.NET et a désormais besoin de graphiques personnalisés — et comment aborder ce module.



Objectif en production

Transformer un Canvas en éditeur : la boucle Modèle, Saisie, État, Rendu, les événements pointeur et tactiles issus de Control, la conversion écran vers monde pour que le hit testing survive au panoramique et au zoom, un état de scène par session plutôt que des champs statiques, l'élimination hors champ, et le moment où une boucle côté client remplace les allers-retours serveur.

1. Le problème en production

Le Module 4 a laissé VisualOperationsStudio avec un bac à sable Canvas : une méthode `DrawPlayground` que l'événement `Redraw` appelle, reconstruisant de zéro chaque tracé, dégradé et transformation. C'est correct, et c'est totalement inerte. Le personnel d'exploitation veut cliquer sur une pompe, la faire glisser à un endroit sensé, se déplacer dans l'usine et zoomer sur un groupe — et une méthode de dessin ne peut répondre à aucune de ces demandes, parce que le canvas du navigateur détient des pixels et que les pixels ignorent quelle forme est une pompe. Dès qu'une surface devient interactive, elle cesse d'être un moteur de rendu et devient une petite machine à états qui, accessoirement, dessine.

2. Concepts clés

Concept	Pourquoi c'est important
Modèle, Saisie, État, Rendu	Quatre couches et une seule règle entre elles : la saisie change l'état, et c'est seulement ensuite qu'une unique méthode de rendu transforme l'état courant en appels de dessin.
Objets de scène	Des classes C# comme <code>NodeModel</code> et <code>EdgeModel</code> qui portent l'identifiant stable, les limites en coordonnées monde, l'état et l'indicateur <code>Selected</code> que les pixels ne peuvent que suggérer.
État de scène par session	La scène appartient à la session d'un seul utilisateur ; une liste <code>static</code> mutable donnerait à chaque visiteur les mêmes nœuds et la même sélection.
Événements de saisie de Control	<code>Canvas</code> dérive de <code>Control</code> : les événements souris (down, move, up), clic, double-clic ainsi que les événements tactiles et gestuels comme <code>TouchStart</code> , <code>Tap</code> , <code>Pinch</code> , <code>Rotate</code> , <code>Swipe</code> et <code>Track</code> sont tous disponibles.
Hit testing fondé sur le modèle	L'API Canvas documentée de Wisej n'expose ni <code>isPointInPath</code> ni <code>isPointInStroke</code> : « sur quoi l'utilisateur a-t-il cliqué ? » se répond donc en comparant un point à une géométrie qui vous appartient.
Conversion écran vers monde	Si le rendu applique <code>écran = monde × zoom + pan</code> , alors chaque coordonnée de pointeur entrante doit repasser par l'inverse avant d'être comparée à quoi que ce soit dans le modèle.
Élimination hors champ	Ignorer les objets de scène dont les limites ne coupent pas le rectangle monde visible, pour qu'une grande topologie ne coûte que ce qui est réellement à l'écran.

3. Le fil de la leçon

Quatre couches et une seule méthode de rendu

Découpez l'éditeur en une couche de modèle (objets métier et de scène), une couche de saisie (souris, pointeur, tactile, clavier et commandes), une couche de transition d'état (sélection, glisser, panoramique, zoom, modifications) et une couche de rendu. La couche de rendu est délibérément ennuyeuse : c'est une fonction déterministe de la scène courante, et elle ne décide de rien. Elle ne cherche pas quelle pompe est sélectionnée ; elle lit `node.Selected` et choisit un remplissage en conséquence. C'est cette contrainte qui permet à la surface de survivre à un redimensionnement, car `Redraw` peut appeler la même méthode à tout instant et récupérer la même image.

La tentation inverse est forte et perd toujours. Un gestionnaire `MouseMove` qui dessine directement un rectangle de mise en évidence est à un redimensionnement, un défilement ou un changement de thème d'une scène qui ne correspond plus à l'idée que l'application se fait du monde. Dans

VisualOperationsStudio, la scène vit dans un objet `TopologyScene` détenu par l'instance de page, si bien que chaque session modifie sa propre topologie, et le `RenderScene` du lab est la seule méthode du fichier autorisée à toucher au canvas.

La saisie vient d'un Control, pas d'un canvas

Comme `Wisej.Web.Canvas` hérite de `Control`, la saisie arrive comme partout ailleurs dans Wisej.NET. `MouseDown` sélectionne et démarre un geste, `MouseMove` déplace le nœud sélectionné ou la vue, et `MouseUp` termine le geste en effaçant l'état de glisser. Chacun de ces gestionnaires suit la même forme en deux lignes : changer l'état, puis appeler la méthode de rendu. Rien ne se dessine à l'intérieur d'un gestionnaire d'événement.

Soyez honnête sur ce que cela coûte. Ce sont des événements côté serveur : chacun est une requête, et un glisser à vitesse humaine en produit un flot. C'est acceptable pour sélectionner et repositionner quelques dizaines de nœuds ; ce ne l'est pas pour de l'encre à main levée ou un mouvement digne d'un jeu. Prévoyez aussi le tactile : les événements gestuels existent, mais les indices d'interface réservés au survol, non ; donnez donc aux poignées une taille suffisante pour être atteintes au pouce, et ne cachez jamais une action essentielle derrière un état de survol.

Le hit testing sur une géométrie qui vous appartient

Le hit testing est une question de modèle. Parcourez la liste des nœuds à rebours — les éléments suivants ont été dessinés par-dessus, l'itération inverse respecte donc l'ordre de profondeur — et renvoyez le premier nœud dont les limites contiennent le point. Les rectangles utilisent `Contains` ; les cercles comparent la distance au carré au rayon au carré, ce qui évite une racine carrée par nœud ; les arêtes en forme de ligne demandent une distance point-segment avec une tolérance, parce que personne ne clique exactement sur une ligne d'un pixel. Pour des tracés réellement complexes, conservez une région de détection simplifiée dans le modèle plutôt que d'essayer de raisonner à rebours à partir des pixels.

Gardez la même discipline pour les utilisateurs du clavier. Les nœuds d'un Canvas ne sont pas des contrôles DOM natifs : rien dans le navigateur ne leur donnera le focus à votre place. Proposez des commandes qui sélectionnent le nœud suivant et déplacent celui qui est sélectionné, et gardez une liste textuelle des mêmes nœuds à côté de la surface. Les deux vues lisent et écrivent l'unique scène, la sélection reste donc cohérente quelle que soit la voie empruntée par l'utilisateur.

Panoramique, zoom et le bug qui ressemble à un bug de sélection

Stockez le panoramique et le zoom dans l'état de la scène et appliquez-les une seule fois, en tête du rendu, encadrés par `Save()` et `Restore()` pour que rien ne fuite vers l'image suivante. La saisie arrive alors dans le mauvais système de coordonnées et doit être reconvertie avant de rencontrer le modèle.

```
private PointF ToWorld(Point screen)
{
    return new PointF((screen.X - _scene.PanX) / _scene.Zoom,
                      (screen.Y - _scene.PanY) / _scene.Zoom);
}

private NodeModel HitTest(Point screen)
{
    PointF p = ToWorld(screen);
    for (int i = _scene.Nodes.Count - 1; i ≥ 0; i--)
        if (_scene.Nodes[i].Bounds.Contains(p))
            return _scene.Nodes[i];
    return null;
}
```

Sautez cette conversion et l'éditeur se comportera parfaitement au zoom 1,0 sans panoramique, puis commencera à rater des nœuds dès que l'un ou l'autre change — un défaut signalé comme « la sélection est cassée après un zoom » et qui n'est qu'une erreur de système de coordonnées. Convertissez en un seul endroit, servez-vous-en pour le hit testing et pour le décalage de glisser, et ajoutez `ToScreen` pour placer une position monde face à une surimpression fixe.

Un budget de redessin, et le point où le client prend le relais

Pour des scènes modérées, préférez un redessin complet et déterministe. Le dessin incrémental paraît moins coûteux et amène avec lui des régions à rafraîchir, la réparation de l'ordre de profondeur, des artefacts d'effacement et la dérive d'état. Lorsque le redessin complet devient réellement coûteux, descendez une échelle fixe avant de changer d'architecture :

- Réduisez le nombre de primitives de la scène, puis éliminez tout ce qui sort du rectangle monde visible.
- Simplifiez la géométrie à faible zoom, et mettez les couches statiques en cache sous forme d'images.
- Séparez les surfaces statiques et dynamiques si la mise en page le permet.
- Seulement ensuite, déplacez vers le client les comportements réellement à haute fréquence.

Laissez `LiveUpdate` désactivé pour le rendu principal. Il pousse les appels de dessin immédiatement, ce qui est utile pour montrer la progression d'une longue construction, mais chaque poussée a un coût et un rendu unique effectué dans le cadre d'une requête de saisie met déjà la scène à jour en une seule réponse. Le même raisonnement condamne la minuterie serveur qui redessine trente ou soixante fois par seconde : elle brûle de la capacité serveur par session et chaque image traverse le réseau. Quand un visuel relève de l'animation plutôt que d'un changement de données, envoyez la nouvelle valeur cible depuis C# et laissez un script client animer vers elle, le serveur restant faisant autorité pour le nombre sémantique. Écrivez noir sur blanc, à partir des mesures du lab, où se situe votre seuil.



Réflexe dessin : la surface d'abord

Choisissez la surface de rendu avant d'écrire le moindre appel de dessin. Demandez-vous où les pixels sont produits et ce qui transite sur le réseau : un gestionnaire Paint de contrôle s'exécute sur le serveur et expédie une image, CellPaint fait de même à l'intérieur d'une cellule de grille, Canvas envoie des commandes de dessin que le navigateur exécute et ne conserve rien après un redimensionnement, et System.Drawing.Managed produit des octets d'image sans aucun contrôle. Gardez le modèle sémantique dans des objets C#, faites de chaque moteur de rendu une fonction déterministe de ce modèle, n'invalidiez que lorsqu'une dépendance visuelle a changé, et laissez toujours un équivalent textuel de tout ce qui compte et n'existe que sous forme de pixels.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Mettre en œuvre le motif Modèle, Saisie, État, Rendu.
- ✓ Utiliser les événements pointeur, souris et tactiles des contrôles Wisej avec une surface Canvas.
- ✓ Mettre en œuvre un hit testing côté serveur pour une interaction à fréquence modérée.
- ✓ Décider quand déplacer la logique d'interaction ou d'animation vers du JavaScript client.
- ✓ Utiliser LiveUpdate, l'invalidation et le redessin sans trafic excessif.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Transformez le bac à sable Canvas en éditeur de topologie pour VisualOperationsStudio. Créez des classes `NodeModel` et `EdgeModel` avec des identifiants stables et une géométrie en coordonnées monde, conservez la scène dans un objet `TopologyScene` stocké par session plutôt que dans un champ statique, et faites de `RenderScene` la seule méthode qui dessine. Ajoutez les conversions `ToWorld` et `ToScreen` pour le décalage de panoramique et le facteur de zoom courants, puis gérez `MouseDown` pour tester en coordonnées monde l'appartenance du point aux limites d'un nœud et positionner `Selected`, `MouseMove` pour faire glisser le nœud sélectionné ou déplacer la vue, et `MouseUp` pour terminer le geste — chacun modifiant l'état puis appelant `RenderScene`. Ajoutez un zoom à la molette ou par bouton centré sur le pointeur, ignorez au rendu les nœuds situés hors du rectangle visible, donnez aux utilisateurs du clavier un moyen de sélectionner et de déplacer un nœud, et relevez le trafic de requêtes que produit un glisser ainsi que le point à partir duquel vous déplacerez la boucle vers du JavaScript client.

Point de départ

VisualOperationsStudio après le Module 4 : la page bac à sable Canvas dessine toute sa scène depuis une méthode `DrawPlayground` appelée par l'événement `Redraw`, avec `Save()` / `Restore()` autour de chaque section transformée, mais chaque forme est codée en dur en coordonnées écran et rien, sur la surface, ne peut être sélectionné, déplacé ou déplacé par panoramique.

Approche suggérée

- ✓ Écrivez `NodeModel` et `EdgeModel` avec un `Id` stable, des `Bounds` en coordonnées monde, un libellé et un indicateur `Selected`, rassemblez-les dans un `TopologyScene` qui porte aussi `PanX`, `PanY` et `Zoom`, et conservez cette scène sur l'instance de page pour que les modifications d'une session n'atteignent jamais une autre.
- ✓ Remplacez `DrawPlayground` par une méthode `RenderScene` qui efface la surface, appelle `Save()`, applique `Translate` et `Scale` depuis la scène, dessine les arêtes puis les nœuds, et appelle `Restore()`; pointez le gestionnaire `Redraw` vers elle et vérifiez qu'un redimensionnement du navigateur ramène la même image.
- ✓ Ajoutez `ToWorld` et `ToScreen`, puis un `HitTest` qui convertit la position du pointeur une seule fois et parcourt la liste des nœuds à rebours pour que le nœud dessiné en dernier l'emporte, en renvoyant `null` pour un espace vide.
- ✓ Câblez `MouseDown`, `MouseMove` et `MouseUp` pour que chacun se contente de modifier l'état de la scène — sélection, décalage de glisser en unités monde, origine du panoramique — puis appelle `RenderScene`; ajoutez le zoom centré sur le pointeur et vérifiez que la sélection tombe encore sur le bon nœud ensuite.
- ✓ Calculez une fois par rendu le rectangle monde visible et ignorez tout ce qui ne le coupe pas, ajoutez des commandes clavier qui sélectionnent le nœud suivant et déplacent celui qui est

sélectionné, puis comptez les requêtes que produit un glisser de deux secondes et consignez les chiffres.

Livrables attendus

- ✓ NodeModel, EdgeModel et un TopologyScene par session, avec des identifiants stables et aucun état statique mutable
- ✓ Conversions ToWorld et ToScreen utilisées pour le hit testing, afin que la sélection reste correcte après un panoramique et un zoom
- ✓ Gestionnaires MouseDown, MouseMove et MouseUp qui modifient l'état puis appellent l'unique méthode RenderScene
- ✓ Élimination hors champ qui ignore les nœuds hors du rectangle visible, plus un parcours clavier pour sélectionner et déplacer un nœud
- ✓ InteractionNotes.md avec le trafic de requêtes mesuré pendant un glisser et le seuil de bascule côté client que vous appliqueriez

Questions de révision

- ✓ Un collègue signale que la sélection fonctionne jusqu'à ce qu'il zoome, après quoi les clics tombent sur la mauvaise pompe ou sur rien. Rien n'a changé dans le code de dessin. Quelle unique conversion manque-t-il, et pourquoi le défaut reste-t-il invisible au zoom 1,0 sans panoramique ?
- ✓ Deux opérateurs ouvrent la topologie en même temps et chacun voit le nœud de l'autre bouger sous son curseur. Dans quoi la scène a-t-elle été stockée, et que cela vous apprend-il sur la place de l'état interactif dans un framework web côté serveur ?
- ✓ Votre glisser produit une requête à chaque mouvement du pointeur. À partir de quelle mesure cesseriez-vous d'optimiser la boucle serveur pour déplacer l'interaction vers du JavaScript client, et qu'est-ce qui devrait malgré tout rester faisant autorité sur le serveur ?



Réflexe dessin : la surface d'abord

Choisissez la surface de rendu avant d'écrire le moindre appel de dessin. Demandez-vous où les pixels sont produits et ce qui transite sur le réseau : un gestionnaire Paint de contrôle s'exécute sur le serveur et expédie une image, CellPaint fait de même à l'intérieur d'une cellule de grille, Canvas envoie des commandes de dessin que le navigateur exécute et ne conserve rien après un redimensionnement, et System.Drawing.Managed produit des octets d'image sans aucun contrôle. Gardez le modèle sémantique dans des objets C#, faites de chaque moteur de rendu une fonction déterministe de ce modèle, n'invalidiez que lorsqu'une dépendance visuelle a changé, et laissez toujours un équivalent textuel de tout ce qui compte et n'existe que sous forme de pixels.