PROGRAMMER SON HOTAS EN DIRECT X

Icewind

8TH DRAGONS

Introduction :

Suite à quelques questions au sujet de la programmation d'un joystick sur Falcon BMS, j'ai décidé de rédiger un tuto pour expliquer une méthode qui fonctionne très bien et que j'utilise.

Comme vous le savez, il y a deux façons de programmer son joystick sur Falcon BMS. La première consiste à utiliser le soft de programmation de votre joystick, comme par exemple T.A.R.G.E.T. pour Thrustmaster, S.S.T. ou encore H.U.D. pour Saitek. Le fonctionnement de ces logiciels est simple, vous assignez le raccourci clavier qui correspond à l'action que vous voulez que le simu effectue au bouton du joystick que vous désirez, via le soft de programmation. Le logiciel se charge donc d'émuler l'appui de la ou des touches afin que le simu réagisse comme si vous effectuiez les commandes au clavier.

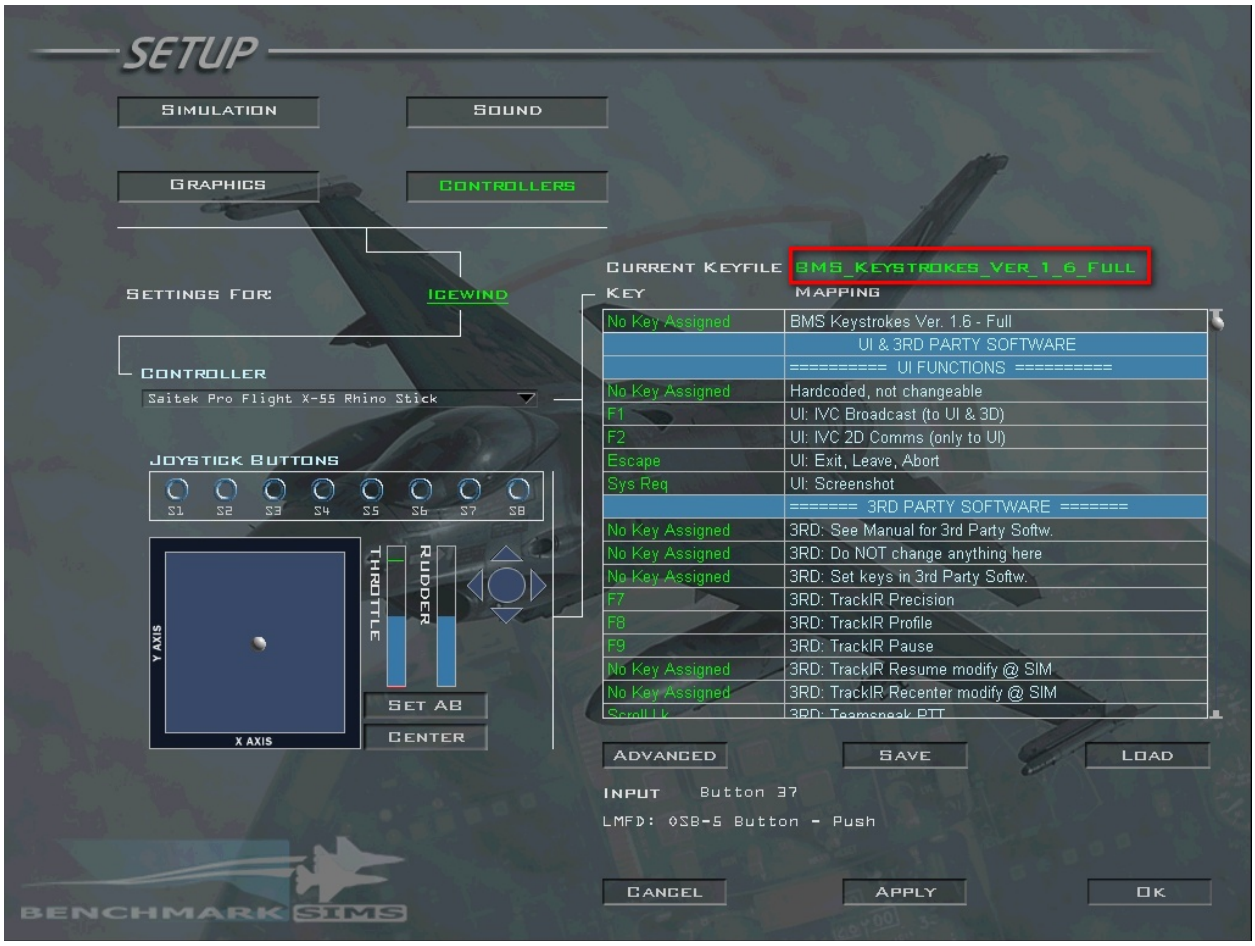
Néanmoins, cette technique comporte plusieurs désavantages. Certains softs (ceux de Saitek par exemple) ont une tendance buguée et vous pouvez vous retrouver avec des appuis de touches parasites provoquant des actions non voulues dans le cockpit, ou des actions qui ne sont pas effectuées alors que vous appuyez sur le bon bouton.

Il existe une méthode qui vous permet de programmer votre joystick et qui utilise les fonctions DirectX. Cette méthode consiste à créer un lien direct entre l'appui du bouton et l'action effectuée dans Falcon BMS. Cela permet de se passer du soft de programmation et donc de ses éventuels bugs. Cependant, il n'y a plus la possibilité d'utiliser plusieurs modes de fonctionnement pour un joystick avec des actions différentes (surcouche) pour chaque bouton selon le mode sélectionné via un rotary ou un bouton à plusieurs positions. L'utilisation de différents modes est possible dans la plupart des softs de programmation des joysticks. En revanche, vous pourrez programmer un bouton de votre joystick comme pinky shift ou dédoubleur, afin de programmer deux actions par bouton. Pour effectuer l'action mise en surcouche du bouton, il suffira de maintenir le pinky shift, et d'appuyer sur le bon bouton.

Il faut savoir que l'interface DirectX supporte 16 contrôleurs de 32 boutons, 8 axes et un POV Hat chacun, ce qui permet d'assigner en tout pas moins de 512 boutons DirectX, 128 axes et 16 POV Hat. C'est donc un puissant moyen de programmation, et programmer en DirectX peut permettre de s'affranchir totalement du soft de programmation.

Pour programmer en DirectX, il y a possibilité de passer par le setup de BMS et d'assigner chaque touche à la bonne action en déroulant le tableau et en cliquant sur la ligne qui nous intéresse, puis en appuyant sur le bouton du joy correspondant. Cependant, j'ai remarqué qu'on perdait parfois toute sa configuration, ce qui avait pour conséquence de nous obliger à la reprendre complètement. Pas trop grave si on a peu de boutons assignés, mais très dérangeant lorsqu'on possède un HOTAS.

La deuxième méthode consiste à utiliser un fichier .key, déjà chargé dans le setup du jeu.



Capture d'écran du menu setup controller avec le nom du keystroke sélectionné

Ces fichiers .key (keystroke.key, BMS.key ou les fichiers .key de Kolbe) se trouvent dans votre répertoire «FalconBMS 4.32\User ». Un fichier .key contient les actions que le simu peut effectuer, et contient les assignations aux raccourcis clavier, ou aux boutons DirectX. Apprendre à le modifier convenablement, afin de sauvegarder sa configuration une fois que tout est programmé comme il faut, permet de ne plus perdre sa config. Le fichier s'ouvre avec le bloc note de Windows, mais je vous conseille d'utiliser le logiciel Notepad++ pour plus de lisibilité.

BMS_Keystrokes_Ver_1_6_Full.key

```
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 0 "BMS Keystrokes Ver. 1.6 - Full"
#=============================================================================================
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 -1 "              UI & 3RD PARTY SOFTWARE"
#=============================================================================================
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 -1 "==========  UI FUNCTIONS  =========="
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 0 "Hardcoded, not changeable"
SimDoNothing -1 0 0X3B 0 0 0 0 "UI: IVC Broadcast (to UI & 3D)"
SimDoNothing -1 0 0X3C 0 0 0 0 "UI: IVC 2D Comms (only to UI)"
SimDoNothing -1 0 0X1 0 0 0 0 "UI: Exit, Leave, Abort"
SimDoNothing -1 0 0xB7 0 0 0 0 "UI: Screenshot"
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 -1 "=======  3RD PARTY SOFTWARE  ======="
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 0 "3RD: See Manual for 3rd Party Softw."
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 0 "3RD: Do NOT change anything here"
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 0 "3RD: Set keys in 3rd Party Softw."
SimDoNothing -1 0 0X41 0 0 0 0 "3RD: TrackIR Precision"
SimDoNothing -1 0 0X42 0 0 0 0 "3RD: TrackIR Profile"
SimDoNothing -1 0 0X43 0 0 0 0 "3RD: TrackIR Pause"
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 0 "3RD: TrackIR Resume modify @ SIM"
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 0 "3RD: TrackIR Recenter modify @ SIM"
SimDoNothing -1 0 0X46 0 0 0 0 "3RD: Teamspeak PTT"
SimDoNothing -1 0 0X37 0 0 0 0 "3RD: Teamspeak Broadcast"
SimDoNothing -1 0 0X37 1 0 0 0 "3RD: Teamspeak Toggle Mike On/Off"
SimDoNothing -1 0 0X37 2 0 0 0 "3RD: Teamspeak Toggle Speaker On/Off"
SimDoNothing -1 0 0X43 0 0 0 0 "3RD: FRAPS Video Capture"
SimDoNothing -1 0 0X44 0 0 0 0 "3RD: FRAPS Screen Capture"
SimDoNothing -1 0 0X57 0 0 0 0 "3RD: FRAPS Benchmarking"
#=============================================================================================
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 -1 "                          LEFT CONSOLE"
#=============================================================================================
SimDoNothing -1 0 0XFFFFFFFF 0 0 0 -1 "===========  TEST PANEL   ==========="
SimOverHeat -1 0 0X3B 5 0 0 1 "TEST: FIRE&OHEAT DETECT Button - Hold"
SimOBOGSBit -1 0 0X3C 5 0 0 1 "TEST: OXY QTY Switch - Hold"
SimMalIndLights -1 0 0X3D 5 0 0 1 "TEST: MAL & IND LTS Button - Hold"
SimMalIndLightsOFF -1 0 0XFFFFFFFF 0 0 0 1 "TEST: MAL & IND LTS Button - Release"
SimProbeHeatMoveUp -1 0 0X3F 5 0 0 1 "TEST: PROBE HEAT Switch - Toggle Up"
SimProbeHeatMoveDown -1 0 0X3E 5 0 0 1 "TEST: PROBE HEAT Switch - Toggle Down"
SimProbeHeatOn -1 0 0XFFFFFFFF 0 0 0 1 "TEST: PROBE HEAT Switch - ON"
SimProbeHeatOff -1 0 0XFFFFFFFF 0 0 0 1 "TEST: PROBE HEAT Switch - OFF"
SimProbeHeatTest -1 0 0XFFFFFFFF 0 0 0 1 "TEST: PROBE HEAT Switch - TEST"
SimEpuGenTest -1 0 0X40 5 0 0 1 "TEST: EPU/GEN Switch - Hold"
SimFlcsPowerTest -1 0 0X41 5 0 0 1 "TEST: FLCS PWR TEST Switch - Hold"
```

Voici ce que ça donne une fois ouvert avec Notepad++

Vous pouvez voir les noms de chacune des fonctions, avec une série de chiffres et de lettres associés, et un commentaire entre guillemets. Nous expliquerons par la suite à quoi correspondent chacune de ces parties.

Connaître son HOTAS

Quelles fonctions sont les plus importantes ? Où allez-vous les programmer ?

Avant d'envisager programmer quoi que ce soit, il faut connaître les fonctions les plus importantes du joystick, et les possibilités de votre matériel. Si vous avez un joystick avec 8 boutons, il est évident que vous n'allez pas pouvoir programmer la totalité des fonctions du HOTAS équipant le F-16. Il faut aussi savoir qu'il existe pour la plupart des joysticks un profil adapté. Pour le Thrustmaster Cougar, il me semble qu'Amraam a déjà créé un profil réaliste. Ne cherchez pas à réinventer la roue si elle existe déjà ! Beaucoup de profils sont à disposition en cherchant un peu, pour la majorité des HOTAS sur le marché. De plus, Mav-jp m'a indiqué à juste titre, que les logiciels Thrustmaster : Foxy et TARGET permettaient la programmation en DirectX. Avec ces logiciels, il n'y a donc pas besoin de passer par un fichier keystroke, vous pouvez programmer votre HOTAS en DirectX via le logiciel. Ainsi, si vous avez déjà cherché des profils et qu'aucun n'est à votre goût ou que votre logiciel ne permet pas la programmation en DirectX, il va falloir mettre les mains dans le cambouis ! Mais rassurez-vous, ce n'est pas si difficile que ça.

La première chose à faire, c'est de bien se renseigner sur le stick du F-16. Je vous conseille les pages 25-26-27 (throttle) et 50-51-52 (stick) du BMS-Dash1 que vous retrouverez dans le répertoire doc du dossier BMS, et ce afin de faire une liste complète de ce dont vous avez besoin. Si vous avez un HOTAS (à partir du x52) vous

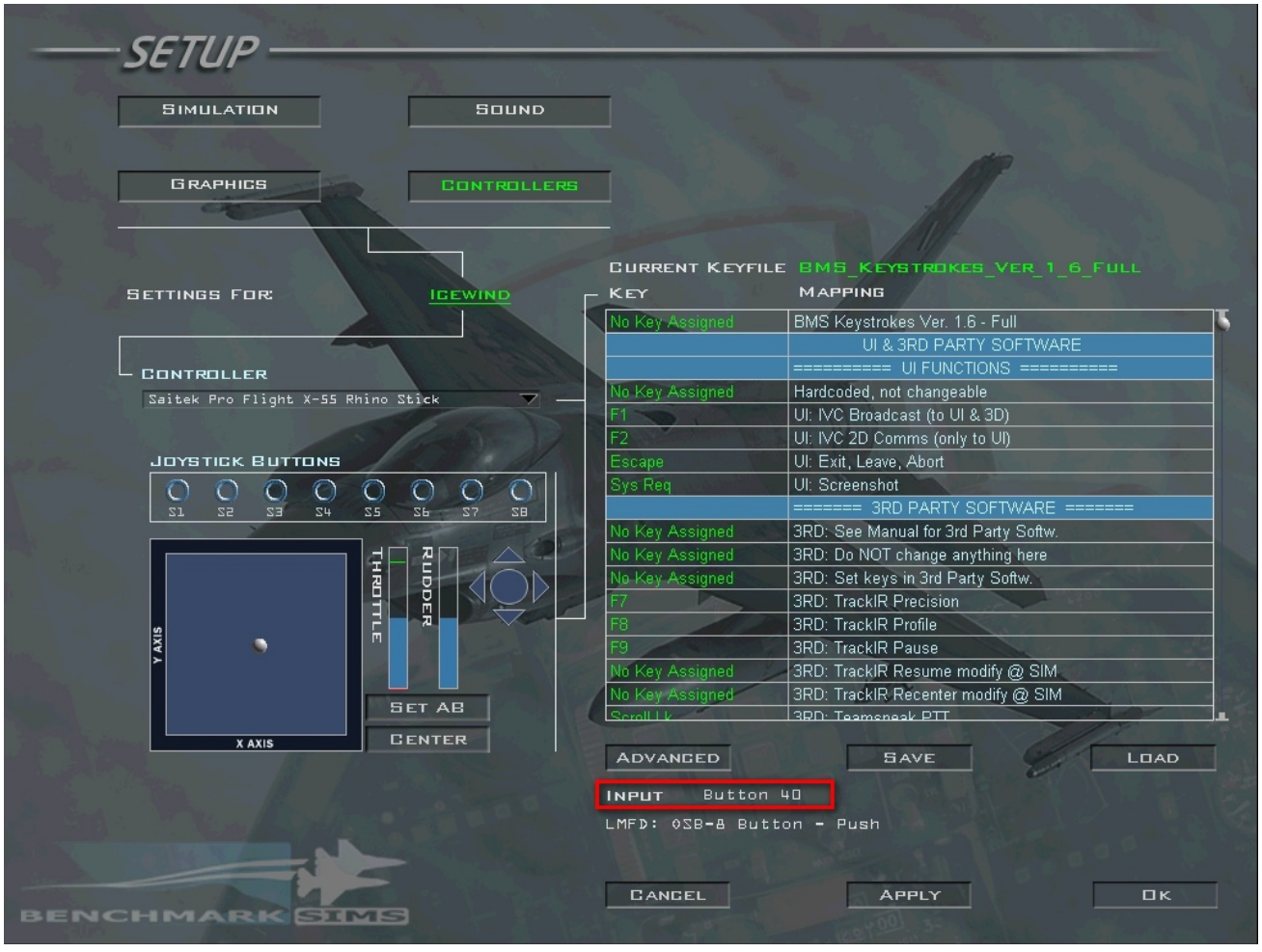
pourrez programmer au moins 75% du HOTAS réel. Sachant que le must reste le Cougar, je suis parvenu à un résultat plus que convenable avec mon x55. Écrivez la liste de ce que vous envisagez programmer, et commencez à réfléchir sur quels boutons ou Hats vous allez caser vos fonctions ! Le but étant de placer les commandes principales sur un accès direct, et placer les fonctions secondaires en « fonctions shiftées ». Là encore, si votre stick possède de nombreux boutons et Hats, le mieux reste d'imiter le HOTAS du F-16, et de le compléter avec d'autres commandes essentielles s'il vous reste de la place, comme le levier de train par exemple.

Faites-vous votre petit schéma sur papier, en assignant chaque fonction à un bouton de votre HOTAS, et soyez prêts à passer à l'étape suivante !

Noter les différents numéros de boutons associés à chaque fonction :

Tout d'abord, il faut choisir le bon fichier .key ! Le plus utilisé est le BMS.key qui convient pour les utilisateurs lambdas. Le fichier keystroke.key est plus reservé aux pitbuilders. Cependant Kolbe nous a offert un superbe travail, et ses différents fichiers keystrokes sont rassemblés dans le dossier « FalconBMS 4.32\User\Joystick\Generic\Kolbe\BMS Keystrokes Files ». Il y a différents fichiers .keys proposés, à vous de choisir celui qui vous paraît le plus adapté à votre matériel et à votre utilisation. Mettez celui retenu de côté, par exemple sur votre bureau et ouvrez le. Les fichiers de Kolbe sont très bien faits car les commandes sont classées par groupe, ce qui permet de les trouver très facilement. Faites tout d'abord une sauvegarde de ce fichier keystroke pour garder l'original quelque part sur votre PC. C'est très important, car à partir de maintenant, vous allez modifier ce fichier.

Ouvrez Falcon BMS, puis allez dans setup, puis Controller. Vous allez appuyer sur chacun des boutons de votre joystick tour à tour, et noter le bouton affiché en dessous du tableau. Prenez votre fiche que vous avez créée et qui regroupe toutes les commandes que vous allez utiliser, et notez le numéro du bouton affiché dans BMS à côté de la fonction que vous avez décidé d'assigner au bouton. Au final, vous devez vous retrouvez avec un numéro de bouton associé à chacune des fonctions qui vous intéresse.



Première méthode : BMS-DirectX Generator de Kolbe

Vous pouvez utiliser le fichier de Kolbe s'intitulant « BMS-DX-Generator 1_6 » et se situant dans votre répertoire « Falcon BMS 4.32\User\Joystick\Generic\Kolbe ».

Comme je n'ai jamais utilisé ce fichier, je ne pourrai malheureusement pas trop vous aider. Si quelqu'un a déjà utilisé le fichier, je suis preneur d'un petit texte explicatif pour insérer dans le tuto. C'est un fichier fait pour modifier automatiquement votre fichier .key pour y insérer exactement les bonnes lignes. Ce fichier a été créé pour éviter de modifier directement le fichier .key via le bloc note, en passant par une interface plus accessible (tableau Excel modifié). A voir selon ce que vous préférez, sachant qu'il y aura certainement des retours de ceux qui ont déjà utilisé ce fichier.

Voici un lien direct vers le post de Checkpoint qui explique ce qu'il faut savoir sur cette méthode, complétant ainsi mon article :
viewtopic.php?p=1445824#p1445824

Un grand merci à lui pour sa contribution !

Deuxième méthode : Éditer à la main son fichier keystroke

Comprendre la mécanique du fichier keystroke :

Décomposons une ligne du fichier keystroke.

BMS_Keystrokes_Ver_1_6_Full.key

```
196 SimSeatOff -1 0 0XFFFFFFFF 0 0 0 1 "SEAT: Safety Lever - SeatArm OFF"
197 SimEject -1 0 0X12 2 0 0 1 "SEAT: EJECT Handle - Hold For Eject"
198 #=============================================================================================
199 SimDoNothing -1 0 0XFFFFFFFF 0 0 0 -1 "====  THROTTLE QUADRANT SYSTEM ======"
200 SimTransmitCom1 -1 0 0XC7 0 0 0 1 "TQS: COMMS Switch Up - UHF"
201 SimTransmitCom2 -1 0 0XCF 0 0 0 1 "TQS: COMMS Switch Down - VHF"
202 SimCommsSwitchLeft -1 0 0XD3 0 0 0 1 "TQS: COMMS Switch Left - IFF OUT"
203 SimCommsSwitchRight -1 0 0XD1 0 0 0 1 "TQS: COMMS Switch Right - IFF IN"
204 SimRangeKnobUp -1 0 0X3C 1 0 0 1 "TQS: MAN RANGE Knob - Up"
```

A gauche en rouge, vous trouverez le nom de la fonction. Au milieu en orange, c'est le code qui permet d'associer la fonction du simu à un raccourci clavier. A droite en jaune, vous trouverez un commentaire qui est en fait un nom plus explicite de la fonction.

Mais cette ligne ne nous intéresse pas réellement car notre but est de faire reconnaître les boutons de notre HOTAS par Falcon, il y a donc une autre façon de coder cela. En fait, seulement la partie surlignée en rouge nous intéressera.

Analysons maintenant une ligne permettant d'associer un bouton du HOTAS à une fonction dans Falcon :

```
1149 SimTriggerSecondDetent 32 8 -2 0 0x0 0
```

A gauche, en rouge, nous retrouvons le nom de la fonction. Au milieu, en orange, c'est le numéro du bouton de notre contrôleur auquel est associée la fonction. Enfin, à droite en jaune, c'est le code qui se retrouve tout le temps à la fin de chaque ligne correspondant à un bouton DirectX, donc cette partie ne changera jamais.

Après avoir choisi le keystroke sur lequel vous allez travailler, il faut impérativement supprimer les lignes à la fin du fichier qui se terminent par « **-1 -2 0 0x0 0** », s'il y en a. Ce sont des boutons déjà programmés, qui ne

correspondent pas forcément à vos besoins et qui ne sont pas forcément placés là où vous le voulez sur votre joystick. Il vaut mieux commencer sa configuration de zéro pour ne pas s'emmêler les pinceaux.

keystrokes.key

```
475  OTWStateStep -1 0 0X39 1 0 0 -2 "Sim-Cycle Object States"
476  OTWToggleAutoScale -1 0 0XB 0 0X2D 2 -2 "Sim-Vehicle Auto-Scaling"
477  OTWSetObjDetail -1 0 0X2 2 0X2D 2 -2 "Set Object Density"
478  OTWSetObjDetail -1 0 0X3 2 0X2D 2 -2 "Set Object Density"
479  OTWSetObjDetail -1 0 0X4 2 0X2D 2 -2 "Set Object Density"
480  OTWSetObjDetail -1 0 0X5 2 0X2D 2 -2 "Set Object Density"
481  OTWSetObjDetail -1 0 0X6 2 0X2D 2 -2 "Set Object Density"
482  OTWToggleBilinearFilter -1 0 0X30 0 0X2D 2 -2 "Sim-Bilinear Filtering"
483  OTWToggleGLOC -1 0 0X30 0 0X2C 2 -2 "Sim-Blackout/Redout"
484  OTWEnterPosition -1 0 0X2E 0 0X2C 2 -2 "Sim-Move To Coordinates"
485  OTWToggleEyeFly -1 0 0X21 0 0X2C 2 -2 "Sim-Testing Function"
486  OTWToggleShading -1 0 0X22 0 0X2D 2 -2 "Sim-Terrain Shading"
487  OTWToggleHaze -1 0 0X23 0 0X2D 2 -2 "Sim-Haze"
488  KevinsFistOfGod -1 0 0XE 2 0 0 -2 "Sim-Fist Of God"
489  OTWToggleLocationDisplay -1 0 0X26 0 0X2C 2 -2 "Sim-World Location"
490  SimStepSMSRight -1 0 0X34 5 0 0 -2 "Sim-Time Of Day"
491  OTWToggleClouds -1 0 0X11 0 0X2D 2 -2 "Sim-Clouds"
492  SimToggleInvincible -1 0 0X2C 0 0X2C 2 -2 "Sim-Invincibility"
493  ReloadTrackIR -1 0 0XFFFFFFFF 0 0 0 1 "Reload TrackIR"
494  RecenterTrackIR -1 0 0XFFFFFFFF 0 0 0 1 "Recenter TrackIR"
495  RecenterJoystick -1 0 0XFFFFFFFF 0 0 0 1 "Recenter Joystick"
496  CampaignQuickSave -1 0 0XFFFFFFFF 0 0 0 1 "Campaign-QuickSave"
497  SimTrigger 0 -1 -2 0 0x0 0
498  SimPickle 1 -1 -2 0 0x0 0
499  SimTriggerFirstDetent 2 -1 -2 0 0x0 0
500  SimDropTrack 3 -1 -2 0 0x0 0
501  SimNextAAWeapon 4 1013 -2 0 0x0 0
502  SimNextAGWeapon 5 1014 -2 0 0x0 0
503  SimNextNavMode 6 1012 -2 0 0x0 0
504  SimFCCSubModeStep 7 -1 -2 0 0x0 0
505  SimCursorUp 8 -1 -2 0 0x0 0
506  SimCursorDown 9 -1 -2 0 0x0 0
507  SimCursorLeft 10 -1 -2 0 0x0 0
508  SimCursorRight 11 -1 -2 0 0x0 0
509  SimToggleMissileCage 12 -1 -2 0 0x0 0
510  SimCommsSwitchLeft 13 -1 -2 0 0x0 0
511  SimCommsSwitchRight 14 -1 -2 0 0x0 0
512  SimTransmitCom1 15 -1 -2 0 0x0 0
513  SimTransmitCom2 16 -1 -2 0 0x0 0
```

Programmer un bouton simple :

Pour programmer un bouton, la démarche est assez simple. Il faut savoir que BMS va coder les boutons à partir du numéro 0. Les boutons de votre contrôleur ont un numéro qui commence à partir de 1, ainsi, il y aura donc un décalage entre le numéro de bouton que vous avez relevé dans le setup controller, et le numéro que vous devrez rentrer sur la ligne du keystroke.

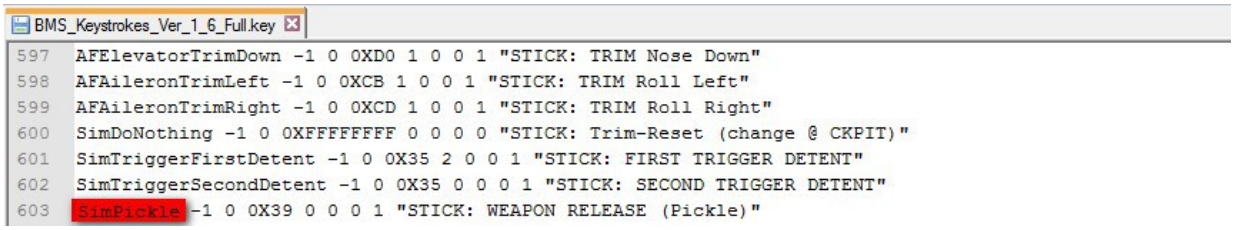
Par exemple, si le numéro de bouton relevé dans BMS est le numéro **1**, le numéro (partie orange surlignée précédemment sur la ligne du fichier keystroke) devra être **0**. (**1-1=0**). On soustrait donc à chaque fois **1** au

numéro affiché dans le setup de BMS, la valeur obtenue sera le nombre que l'on devra rentrer dans le keystroke.

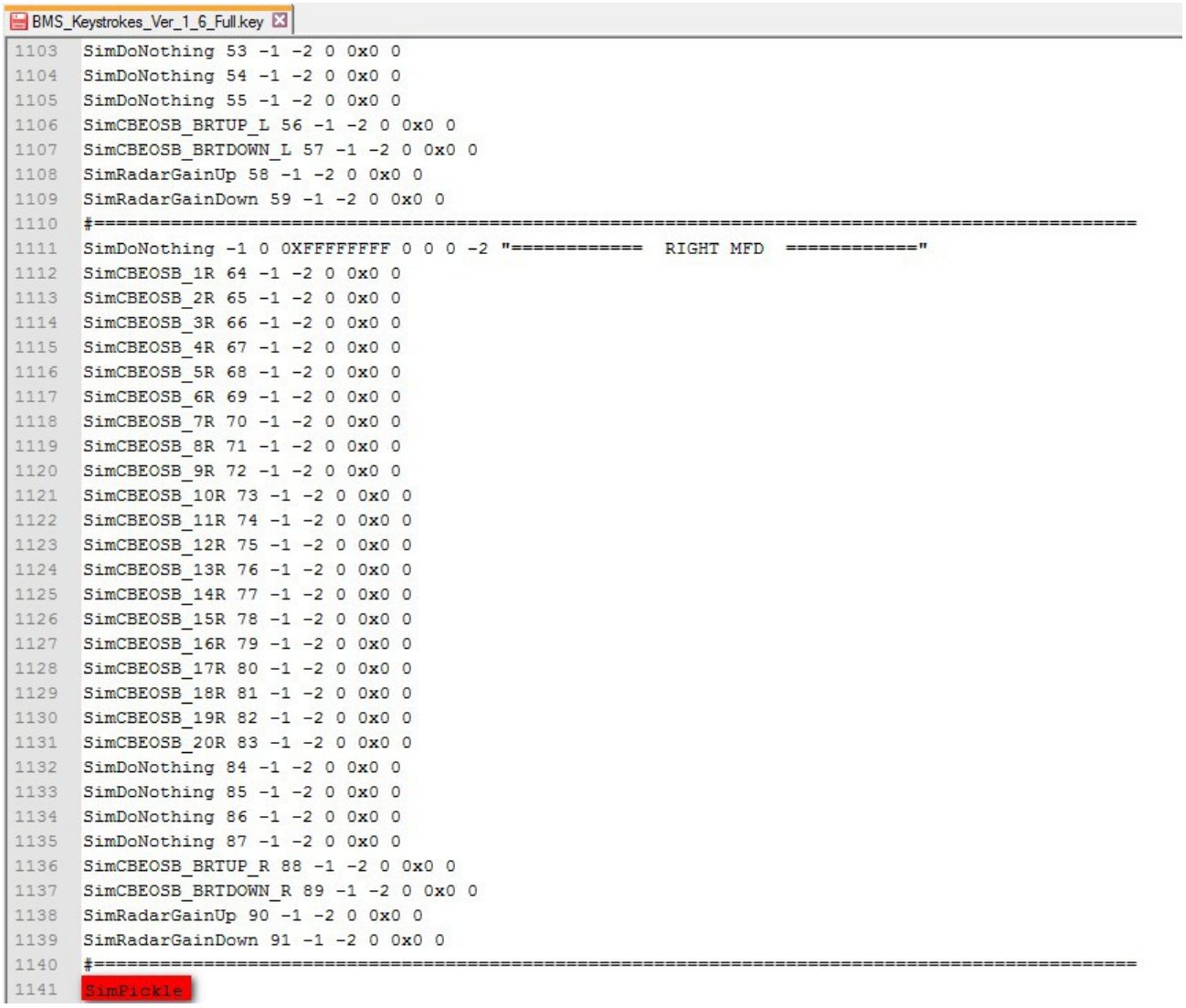
La méthode est donc simple. Vous savez quelles fonctions vous voulez assigner et à quel endroit sur votre joystick. Vous savez aussi quel est le numéro du bouton de votre joystick qui vous intéresse car BMS vous l'a donné, et vous l'avez noté sur votre fiche récapitulative des fonctions que vous allez programmer sur votre HOTAS.

La première chose à faire, c'est de chercher la ligne correspondant à la fonction que l'on veut programmer. Pour cela, il faut parcourir le keystroke. Nous allons programmer le bouton Pickle sur notre joystick. Je cherche la fonction qui correspond au bouton Pickle dans mon fichier keystroke.

J'ai choisi le « **BMS_Keystrokes_Ver_1_6_Full.key** ».



Je trouve la fonction sous le nom de « **SimPickle** ». Je sélectionne uniquement la partie en rouge, et je la copie-colle en bas de mon fichier. Voici ce que j'obtiens :



Je vais maintenant ajouter le restant de la ligne que voici : « **0 -1 -2 0 0x0 0** » Le premier chiffre (en rouge) correspond au numéro de bouton. On le laisse à **0** provisoirement, et on ne touche pas au reste de la ligne (partie jaune). On obtient donc ceci :

```
1141  SimPickle 0 8 -2 0 0x0 0
```

Nous devons maintenant associer le bon numéro de bouton. Comme je n'ai pas fait de feuille pour me repérer et que je n'ai qu'un bouton à programmer, je vais aller chercher le numéro de mon bouton dans Falcon, dans le menu setup controller. Une fois sur la bonne page, j'appuie sur le bouton de mon joystick auquel je veux affecter la fonction, et le numéro du bouton apparaît en bas du tableau.

Mon futur bouton Pickle a donc **34** comme numéro de bouton. On corrige le décalage : **34-1=33**. Notre bouton portera donc le numéro **33** dans le fichier keystroke. On remplace le chiffre **0** par le numéro du bouton calculé. On remplace donc le **0** par **33** dans notre cas.

```
1141  SimPickle 33 8 -2 0 0x0 0
```

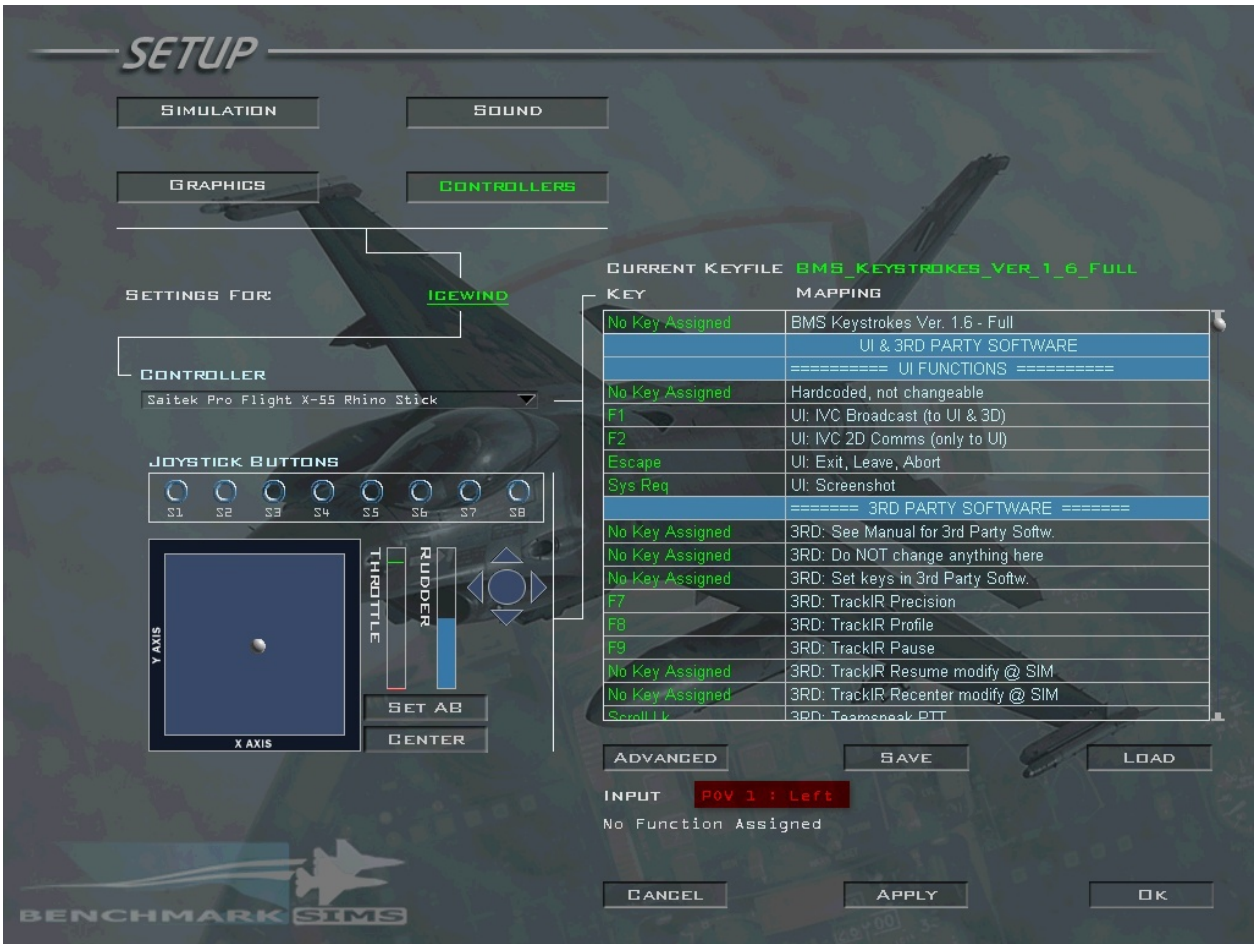
On enregistre le fichier keystroke et on le ferme. On retourne dans Falcon, dans le menu setup controller et on reload le fichier keystroke (bouton load du menu setup controller). On teste le bouton en appuyant dessus.

On vérifie que la bonne fonction est assignée. J'ai bien la fonction pickle reconnue en bas du tableau donc c'est tout bon.

Si vous avez réussi à programmer correctement un bouton, c'est que vous avez compris le principe. Vous pouvez donc rentrer toutes les lignes adéquates dans le keystroke en reprenant les étapes de cette partie une par une, mais vous ne serez pas obligés d'enregistrer votre fichier pour chaque ligne ajoutée afin de vérifier si ça fonctionne. Vous ne le ferez qu'une fois le fichier terminé, pour aller tester si tout va bien. Attention, je n'ai abordé ici que les boutons simples ! Je vais maintenant parler d'un cas particulier, qui concerne les POV Hats avant de parler de la programmation des fonctions shiftées.

Programmer un POV hat :

Un POV Hat, est un Hat reconnu différemment par Windows, et donc par BMS. La méthode de programmation change donc légèrement. Votre Hat est un POV Hat s'il est reconnu en tant que tel dans le menu setup controller de BMS.



Vous remarquez qu'il n'y a plus marqué « button » en dessous du tableau, mais « POV » suivi du numéro du POV Hat et de la direction dans laquelle vous l'avez orienté lorsque vous l'avez bougé.

Nous allons maintenant chercher à comprendre comment on programme un POV Hat en DirectX. Décomposons donc une ligne de programmation d'un POV Hat dans le fichier keystroke :

```
1142 SimCMSUp 0 -1 -3 0 0x0 0
```

Nous retrouvons en rouge le nom de la fonction. En orange, nous avons le numéro du POV Hat. La partie jaune contient le code qui ne change jamais. Vous remarquerez qu'il y a une partie supplémentaire qui est ici en vert. Ce numéro représente l'orientation du Hat. En effet, comme la plupart des Hats ont 4 à 8 positions, Falcon BMS doit savoir quelle fonction va avec quelle direction. Ce petit schéma donne les numéros associés aux directions du Hat :

Source du schéma : http://www.benchmarksims.org/forum/atta ... 1269697975

Je vais vous montrer comment programmer un POV Hat, avec l'exemple du Hat CMS. Tout d'abord, je vais chercher le nom des fonctions correspondant à celles du Hat CMS, et je copie uniquement la partie en rouge de la ligne, je colle cette partie en bas de mon fichier keystroke. Le Hat CMS du F-16 possède 4 directions, et chacune est associée à une fonction. Je me retrouve avec quatre lignes collées en bas de mon fichier.

```
1141  SimPickle 33 -1 -2 0 0x0 0
1142  SimCMSUp
1143  SimCMSDown
1144  SimCmsLeft
1145  SimCMSRight
```

Ensuite, je rajoute le reste de la ligne de code nécessaire : « **0 -1 -3 0 0x0 0** »
Le premier numéro correspond toujours au numéro du POV (rouge), et le 4ème numéro (orange) correspond à la direction du Hat. Pour l'instant, le numéro orange correspondant à la direction du Hat est à **0**, et le numéro du Hat aussi.

```
1141  SimPickle 33 -1 -2 0 0x0 0
1142  SimCMSUp 0 -1 -3 0 0x0 0
1143  SimCMSRight 0 -1 -3 0 0x0 0
1144  SimCMSDown 0 -1 -3 0 0x0 0
1145  SimCmsLeft 0 -1 -3 0 0x0 0
```

Comme mon POV Hat est reconnu comme numéro **1** (premier screen), je corrige le décalage en soustrayant 1 au numéro, et je me retrouve avec 1-1=0, je laisse donc **0** comme numéro du POV Hat dans le fichier keystroke. Maintenant occupons-nous de la direction du Hat. CMS Up correspond à la direction vers le haut du Hat, donc selon le schéma, le numéro associé à sa direction sera **0**.

Selon le schéma trois screens plus haut, nous devons attribuer les numéros suivants :

- **CMS Up : 0 (haut)**
- **CMS Right : 2 (droite)**
- **CMS Down : 4 (bas)**
- **CMS Left : 6 (gauche)**

On remplace donc dans chacune des lignes le numéro associé à la direction par le numéro adéquat selon la direction du Hat voulue. Voici ce que l'on obtient (j'ai classé les lignes pour que les numéros soient croissants) :

```
1141  SimPickle 33 -1 -2 0 0x0 0
1142  SimCMSUp 0 -1 -3 0 0x0 0
1143  SimCMSRight 0 -1 -3 2 0x0 0
1144  SimCMSDown 0 -1 -3 4 0x0 0
1145  SimCmsLeft 0 -1 -3 6 0x0 0
```

Vous pouvez donc enregistrer le fichier keystroke et effectuer vos tests dans le menu de BMS, en reloadant votre fichier keystroke.

En bas du tableau du menu setup controller, lorsque vous bougez votre Hat dans chacune des positions, vous devez avoir :

- **CMS Up pour "POV 1 : Up"**
- **CMS Right pour "POV 1 : Right"**
- **CMS Down pour "POV 1 : Down"**
- **CMS Left pour "POV 1 : Left"**

Il faut savoir qu'un POV Hat non programmé est attribué par défaut à la fonction « View Panning » qui permet de bouger la tête dans le cockpit.

On peut donc passer aux fonctions shiftées !

Programmation de fonctions shiftées

Programmer un bouton shifté :

Une fonction shiftée est une fonction supplémentaire ajoutée à un bouton. Par exemple, si je décide que mon bouton 34 qui sert pour la fonction Pickle en temps normal, serve pour activer le mode dogfight lorsque je maintiens enfoncé un bouton et que j'appuie sur mon bouton 34, c'est programmable dans le keystroke. Il faut choisir un bouton shift, qui devra être maintenu enfoncé en plus du bouton qui enclenchera l'action voulue. On assigne souvent au bouton qui tombe sous le petit doigt (sur le joystick) cette fonction de shifteur.

En résumé, si j'appuie sur mon bouton 34, le simu exécutera l'action Pickle. Par contre, si je maintiens le bouton shift (bouton sous le petit doigt) et que j'appuie sur le bouton 34, le mode dogfight sera activé, mais l'action Pickle ne sera pas effectuée.

La première modification devra être faite dans le fichier « **falcon bms.cfg** », se trouvant dans votre répertoire « Falcon BMS 4.32\User\Config ». Il faut ouvrir le fichier, et vérifier que la ligne suivante se trouve bien dans le fichier : **set g_nHotasPinkyShiftMagnitude 256**

falcon bms.cfg

```
73  ////////////////////
74  // Misc Settings //
75  ////////////////////
76
77  set g_bVoiceCom 1                          // Enable Voice Communications
78  set g_nF1TeamUiFreq 307300                 // Frequency for F1 UI voice comms (for Force on Force type missions).
79  set g_nF2TeamUiFreq 1234                   // Frequency for F2 UI voice comms (for Force on Force type missions).
80  set g_bhudAOA 1                            // Turns OFF the HUD AOA.
81  set g_bLocalEnvironmentalDate 0            // Makes the TimeManager use the local system date for environmental lighting calculations
82  set g_bHotasDgftSelfCancel 0               // set to 1 = SRM and MRM override callbacks will call the override cancel callback for you
83  set g_nHotasPinkyShiftMagnitude 256        // Enable DirectX Shifting and specify button offset
84  set g_nHotasShiftQuickPressTimeLimit 200
85  set g_fFOVIncrement 5                      // Sets how much the field of view should change for each keypress in degrees
86  set g_fMaximumFOV 80                       // Limits the maximum amount that the FOV can be increased
87  set g_bNoAAAEventRecords 0                 // This option, set to 1, removes AAA shots from debrief
88  set g_bACMIRecordMsgOff 1                  // Turns off the ACMI RECORDING msg
89  set g_nPadlockBoxThickness 1               // Determines the thickness of the padlock box
90  set g_nDynamicVoices 16                    // Maximum number of voices allocated by the sound code
91  set g_nSoundUpdateMS 10                    // How many milliseconds must elapse before the sound code updates
92  set g_nSoundSwitchFix 1                    // This patch may fix the problem of AI comms disappearing
93  set g_bPilotEntertainment 0                // Enables the user to control WinAmp
94  set g_nWinAmpInitVolume 204                // Sets the initial WimAmp volume
95  set g_fAmbientmin 0.1                      // Allows to tweak sky brightness at night
96  set g_bForce16bitDisplay 0                 // Forces 16 bit display device.
97  set g_bNewThreadTiming 1                   // If experiencing hiccups/stuttering in the UI with multi-core processors, try set to 0
```

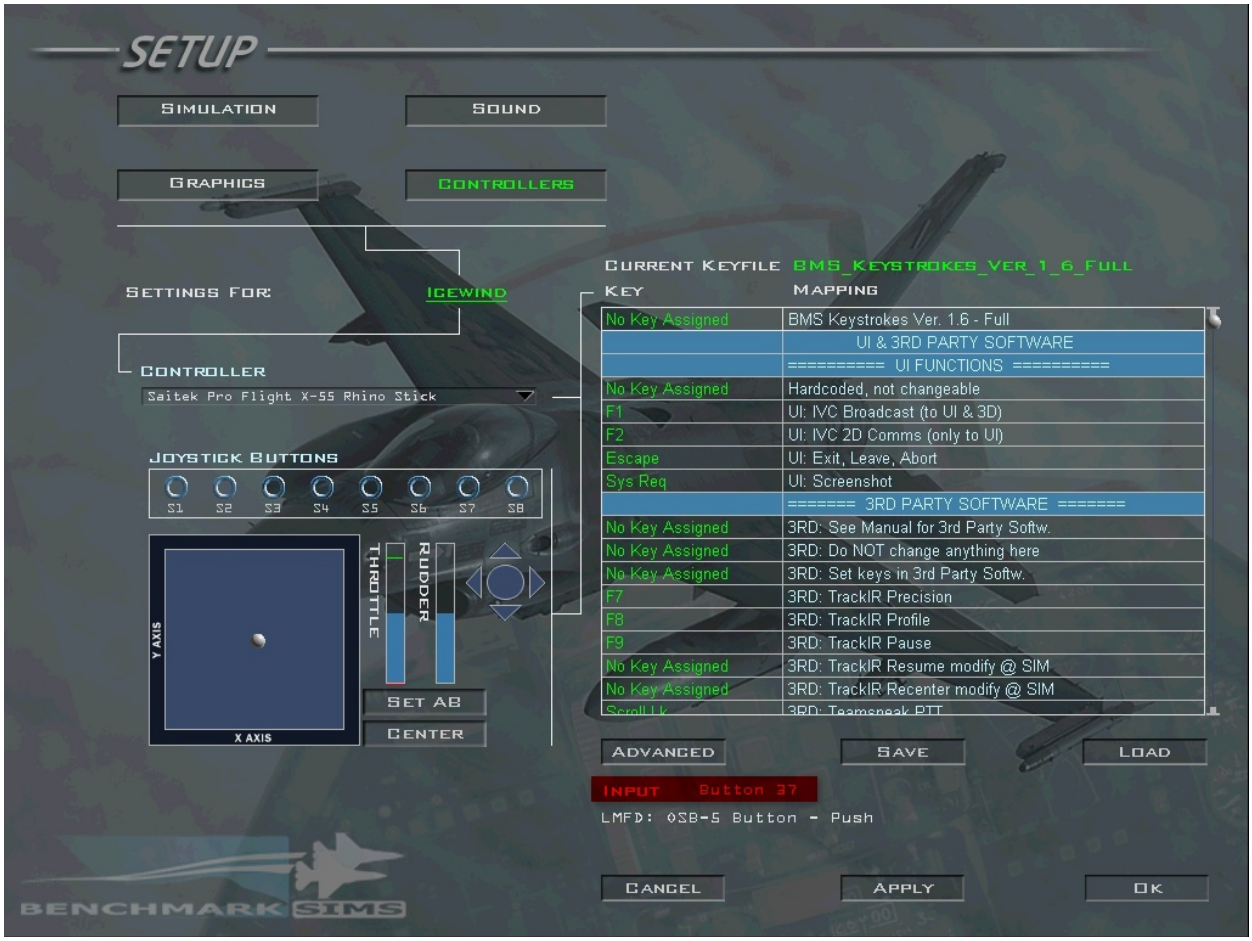
Cette ligne active la possibilité d'utiliser les fonctions shiftées. Nous savons qu'un contrôleur permet d'utiliser 32 boutons DirectX. Donc le numéro de bouton rentré dans le keyfile si celui-ci appartient au premier contrôleur se situera entre 0 et 31. Pour le deuxième contrôleur, ce sera entre 32 et 63, etc...

Le nombre situé à la fin de la ligne définit le début des plages de numéros qui seront réservées dans le keystroke pour les fonctions shiftées. Lorsque le numéro est à 256, cela signifie que les fonctions shiftées porteront au minimum le numéro 256 dans le fichier keystroke, et que les plages de numéros entre 0 et 255 sont libres pour les autres contrôleurs. Je vous conseille de laisser la valeur 256, car cela laisse la place pour 8 contrôleurs (largement suffisant normalement), et cette valeur est un multiple de 32 (ce qui évite de chevaucher 2 plages de numéros de boutons). Enregistrez le fichier **falconbms.cfg** et quittez.

Pour programmer des fonctions shiftées, il va falloir encore une fois comprendre la mécanique de codage. Vous allez devoir choisir un bouton qui fait office de shifteur. C'est celui que vous devrez maintenir enfoncé pour activer les fonctions shiftées. La ligne définissant ce bouton est « **SimHotasPinkyShift 0 8 -2 0 0x0 0** ». Vous devez rentrer cette ligne dans votre fichier keystroke comme d'habitude. La partie rouge désigne le nom de la fonction, la partie orange désigne le numéro du bouton, et la partie jaune ne doit pas changer.

```
1146 SimHotasPinkyShift 0 8 -2 0 0x0 0
```

Quand je suis dans le menu setup controller de BMS et que j'appuie sur le bouton que j'ai choisi comme shifteur, BMS m'affiche le numéro 37 :



Je corrige le décalage en soustrayant 1 à cette valeur (**37-1=36**) donc je rentre 36 dans la partie orange.

```
1146 SimHotasPinkyShift 36 8 -2 0 0x0 0
```

Comme on a rentré **256** dans le fichier **falconbms.cfg**, il faut rajouter la ligne suivante : « **SimHotasPinkyShift 256 -1 -2 0 0x0 0** ».

```
1141 SimPickle 33 8 -2 0 0x0 0
1142 SimCMSUp 0 -1 -3 0 0x0 0
1143 SimCMSRight 0 -1 -3 2 0x0 0
1144 SimCMSDown 0 -1 -3 4 0x0 0
1145 SimCmsLeft 0 -1 -3 6 0x0 0
1146 SimHotasPinkyShift 36 8 -2 0 0x0 0
1147 SimHotasPinkyShift 256 8 -2 0 0x0 0
```

Elle doit absolument être présente, car son absence provoquera un dysfonctionnement des fonctions shiftées. Le bouton shift ne serait pas relâché. Mais ce 256 doit changer en fonction du numéro que l'on a rentré pour l'autre « **SimHotasPinkyShift** ». J'avais rentré 36 comme numéro. Pour savoir quelle valeur rentrer dans la zone orange, on applique **256 + numéro bouton du premier SimHotasPinkyShift**, on a donc ici **256 + 36 = 292**. Je rentre donc 292 dans la partie orange.

```
1141  SimPickle 33 8 -2 0 0x0 0
1142  SimCMSUp 0 -1 -3 0 0x0 0
1143  SimCMSRight 0 -1 -3 2 0x0 0
1144  SimCMSDown 0 -1 -3 4 0x0 0
1145  SimCmsLeft 0 -1 -3 6 0x0 0
1146  SimHotasPinkyShift 36 8 -2 0 0x0 0
1147  SimHotasPinkyShift 292 8 -2 0 0x0 0
```

Pour programmer un bouton en fonction shiftée c'est très simple si vous n'avez eu aucun problème jusqu'ici ! Vous faites comme d'habitude, vous copiez le nom de la fonction à la fin de votre fichier keystroke, vous y rajoutez le code de d'habitude, à savoir : « **0 8 -2 0 0x0 0** » Ici on s'intéresse à la fonction dogfight qui porte le nom de « **SimSelectSRMOverride** ». Je veux programmer mon bouton en surcouche du bouton Pickle, qui porte le numéro **33** dans le keystroke. Cette fois-ci, le seul truc qui change, c'est qu'on ne rentre pas ce numéro là, mais la valeur qu'on obtient lorsqu'on lui additionne **256**. Voici donc le calcul pour obtenir le numéro de la fonction shiftée : **33+256=289**. Je rentre donc **289** dans la partie orange. Attention, je pars du principe que je connais déjà le numéro que porte le bouton Pickle dans le keystroke (je lui ai déjà soustrait 1). Si vous cherchez le numéro du bouton directement dans BMS, n'oubliez pas de lui soustraire **1** avant de lui ajouter **256**, car sinon ça ne fonctionnera pas !

```
1141  SimPickle 33 8 -2 0 0x0 0
1142  SimCMSUp 0 -1 -3 0 0x0 0
1143  SimCMSRight 0 -1 -3 2 0x0 0
1144  SimCMSDown 0 -1 -3 4 0x0 0
1145  SimCmsLeft 0 -1 -3 6 0x0 0
1146  SimHotasPinkyShift 36 8 -2 0 0x0 0
1147  SimHotasPinkyShift 292 8 -2 0 0x0 0
1148  SimSelectSRMOverride 289 8 -2 0 0x0 0
```

Le souci, c'est qu'on ne peut plus vérifier le bon fonctionnement d'une fonction shiftée dans le setup de BMS. On est obligé de lancer un vol, pour vérifier si le changement est bien pris en compte. Dans mon cas, en vol, je maintiens mon PinkyShitch, et j'appuie sur mon bouton Pickle en même temps. Je passe en mode dogfight.

Vous devez donc avoir deux lignes « **SimHotasPinkyShift** » dans votre fichier keystroke pour que les fonctions shiftées fonctionnent, j'ai donc rentré dans mon cas les lignes suivantes :

SimHotasPinkyShift 36 8 -2 0 0x0 0
SimHotasPinkyShift 292 8 -2 0 0x0 0

Sachez que le **Sim Hotas Pinky Shift** comporte également la fonction **Sim Hotas Pinky Switch**. Sur le stick réel du F-16, le bouton qui tombe sous le petit doigt a une fonction spécifique et n'est pas un shifteur. Ainsi, dans BMS, une fonction lui correspond : **Sim Hotas Pinky Switch**. On pourrait se demander comment programmer à la fois la fonction de shifteur, et la fonction réelle de Pinky Switch. BMS a bien fait les choses, et la fonction **SimHotasPinkyShift** fait les deux à la fois. Ainsi, si votre appui est court sur le bouton, il agira comme le Pinky Switch. Si l'appui sur le bouton est prolongé, il agira comme shifteur, et l'action de Pinky Switch ne sera pas effectuée. Vous pouvez modifier le temps qui définit ce qui est long ou court dans le fichier **falconbms.cfg** (« Falcon BMS 4.32\User\Config »), en utilisant cette ligne : **set g_nHotasShiftQuickPressTimeLimit n**
La valeur de **n** est **200** par défaut, ce qui signifie que si vous appuyez moins de 200 millisecondes, l'appui sera considéré comme un appui court, au-delà d'un appui de 200 millisecondes, le bouton sera utilisé comme shifteur car l'appui sera considéré comme un appui long.

falcon bms.cfg

```
73  ////////////////////
74  // Misc Settings //
75  ////////////////////
76
77  set g_bVoiceCom 1                          // Enable Voice Communications
78  set g_nF1TeamUiFreq 307300                 // Frequency for F1 UI voice comms (for Force on Force type missions).
79  set g_nF2TeamUiFreq 1234                   // Frequency for F2 UI voice comms (for Force on Force type missions).
80  set g_bhudAOA 1                            // Turns OFF the HUD AOA.
81  set g_bLocalEnvironmentalDate 0            // Makes the TimeManager use the local system date for environmental lighting calculations
82  set g_bHotasDgftSelfCancel 0               // set to 1 = SRM and MRM override callbacks will call the override cancel callback for you
83  set g_nHotasPinkyShiftMagnitude 256        // Enable DirectX Shifting and specify button offset
84  set g_nHotasShiftQuickPressTimeLimit 200
85  set g_fFOVIncrement 5                      // Sets how much the field of view should change for each keypress in degrees
86  set g_fMaximumFOV 80                       // Limits the maximum amount that the FOV can be increased
87  set g_bNoAAAEventRecords 0                 // This option, set to 1, removes AAA shots from debrief
88  set g_bACMIRecordMsgOff 1                  // Turns off the ACMI RECORDING msg
89  set g_nPadlockBoxThickness 1               // Determines the thickness of the padlock box
90  set g_nDynamicVoices 16                    // Maximum number of voices allocated by the sound code
91  set g_nSoundUpdateMS 10                    // How many milliseconds must elapse before the sound code updates
92  set g_nSoundSwitchFix 1                    // This patch may fix the problem of AI comms disappearing
93  set g_bPilotEntertainment 0                // Enables the user to control WinAmp
94  set g_nWinAmpInitVolume 204                // Sets the initial WimAmp volume
95  set g_fAmbientmin 0.1                      // Allows to tweak sky brightness at night
96  set g_bForce16bitDisplay 0                 // Forces 16 bit display device.
97  set g_bNewThreadTiming 1                   // If experiencing hiccups/stuttering in the UI with multi-core processors, try set to 0
```

Enfin, sachez que si votre HOTAS est considéré comme deux périphériques différents (throttle et stick séparés), ou que vous utilisez plusieurs périphériques distincts comme des MFD en plus de votre joystick par exemple, vous pouvez n'utiliser qu'un seul et même bouton shifteur (sur votre HOTAS par exemple) qui contrôlera la surcouche de tous vos périphériques. Si vous attribuez une fonction shiftée à un bouton de l'un de vos MFD, que vous maintenez votre shifteur, puis que vous appuyez sur le bouton du MFD, la fonction secondaire sera effectuée à la place de la fonction primaire. Il n'y a aucun changement dans la manière de programmer, il suffit juste d'appliquer ce qui est écrit plus haut !

Voilà, vous savez comment programmer un shifteur et un bouton shifté. Il ne vous reste plus qu'à apprendre comment programmer un POV Hat shifté ! (Mais qu'est-ce qu'il est long son tuto à c'lui là...)

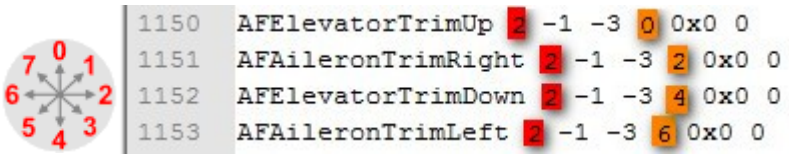
Programmer un POV Hat shifté :

Nous arrivons à la fin du tuto, et heureusement pour vous ! Programmer un POV Hat en fonction shiftée n'est pas très difficile. On prend le numéro du POV Hat que l'on a rentré dans le fichier keystroke, qui est **0** dans mon cas. Pour un POV hat, au lieu de rajouter **256** au numéro du Hat, on rajoute toujours **2**. Ainsi, on a **0+2=2**. Si le numéro du POV Hat avait été **1**, on aurait rentré **3** comme numéro.

Je vais programmer le Trim en surcouche de mon hat CMS. Je cherche le nom des fonctions que je copie-colle en bas de mon keystroke. Je rajoute le code adéquat. « **0 8 -2 0 0x0 0** ».

```
1141  SimPickle 33 8 -2 0 0x0 0
1142  SimCMSUp 0 -1 -3 0 0x0 0
1143  SimCMSRight 0 -1 -3 2 0x0 0
1144  SimCMSDown 0 -1 -3 4 0x0 0
1145  SimCmsLeft 0 -1 -3 6 0x0 0
1146  SimHotasPinkyShift 36 8 -2 0 0x0 0
1147  SimHotasPinkyShift 292 8 -2 0 0x0 0
1148  SimSelectSRMOverride 84 8 -2 0 0x0 0
1149  SimTriggerSecondDetent 32 8 -2 0 0x0 0
1150  AFElevatorUp
1151  AFElevatorDown
1152  AFAileronLeft
1153  AFAileronRight
```

Je remplace la partie rouge par le numéro calculé, à savoir **2**, et je remplace la partie orange par le numéro correspondant à la position physique du Hat lorsque l'on effectue l'action.



La démarche est donc très proche de celle applicable aux boutons.

Informations complémentaires

Une ligne qui peut se montrer utile : SimDoNothing

Certains boutons ou Hats ont une fonction prédéfinie dans le simulateur lorsque l'on crée un profil DirectX. Un POV Hat a par défaut le rôle de « View Panning », c'est lui qui permet de bouger la tête du pilote dans le pit si vous n'avez pas de track IR. Si vous n'êtes pas possesseur d'un track IR et que vous n'utilisez pas un autre système de track, la fonction primaire du Hat sera de bouger la vue. Ce rôle est effectué, même si vous appuyez sur votre shifteur dans le cas où vous n'avez programmé aucune fonction secondaire au Hat.

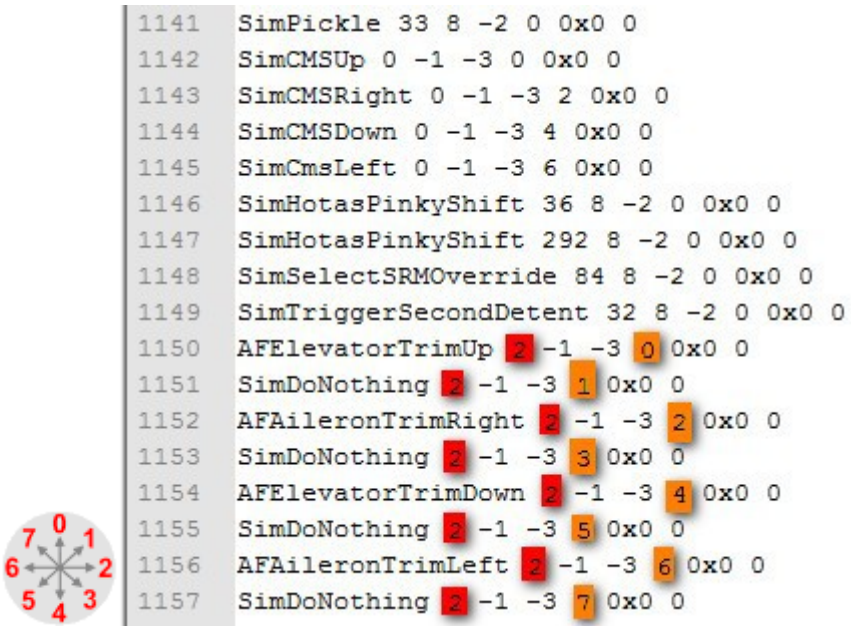
Prenons le cas où vous utiliseriez le Hat pour bouger la vue dans votre cockpit en fonction primaire, et pour trimmer votre avion comme fonction secondaire. Toutes les directions du Hat sont utilisées en fonction primaire, et seulement quatre directions sont utilisées en fonction secondaire (haut/bas/droite/gauche). Les coins, eux restent inutilisés en fonction secondaire, et gardent donc leur fonction primaire : bouger la vue. Ainsi, en vol, si vous appuyez sur votre shifteur et que vous trimmez votre avion, le fait de déplacer le Hat dans un des coins provoquera un déplacement involontaire de la caméra. Pour remédier à ça, une ligne permet d'attribuer une fonction : celle de ne rien faire.

Ainsi, cette ligne vous permettra de ne pas être parasité par des actions du simulateur effectuées contre votre volonté.

Cette ligne peut fonctionner pour un Hat ou un bouton. Je veux neutraliser le mouvement de la caméra lorsque j'oriente mon Hat dans les coins, alors que j'appuie sur le shifteur pour l'utiliser en tant que Trim. Voici comment je dois procéder :

Je copie à la fin de mon fichier la fonction « **SimDoNothing** ». Si je veux l'utiliser pour un Hat, je rajoute « **0 -1 -3 0 0x0 0** », si je veux l'utiliser pour un bouton, je rajoute « **0 8 -2 0 0x0 0** ». Ensuite je change le numéro du bouton ou du Hat ainsi que le numéro de la position du Hat que je veux neutraliser (dans le cas du Hat), comme nous l'avons déjà vu auparavant.

Voici ce que je dois obtenir dans l'exemple du Hat utilisé pour la vue et pour le trim.



On a neutralisé les positions **1,3,5,7** du schéma lorsque le shifteur est maintenu, donc les coins du Hat, qui ne sont pas programmés pour les Trims. Ainsi, lorsque l'on appuiera sur le shifteur et que l'on bougera le Hat dans les coins, la caméra ne sera pas déplacée.

Pour neutraliser un bouton, on applique la même méthode mais avec les spécificités de code propres au bouton. Si je veux neutraliser le bouton que j'avais programmé pour la fonction Pickle, je supprime la ligne :

```
1141  SimPickle 33 8 -2 0 0x0 0
```

Et j'ajoute la ligne suivante :

```
1141  SimDoNothing 33 8 -2 0 0x0 0
```

Maintenant, je vous conseille de n'utiliser cette fonction que lorsqu'elle est nécessaire, ne vous amusez pas à l'utiliser pour chaque bouton qui n'a pas de fonction car c'est totalement inutile !

Spécificité du Thrustmaster Hotas Cougar

Sur certains joysticks comme le Cougar, il y a besoin d'une ligne spécifique pour programmer le dogfight switch. En effet, le bouton possède trois positions physiques mais seulement deux sont reconnues comme un bouton. Ainsi, si vous passez en mode dogfight et que vous revenez en position centrale, vous ne pourrez pas coder le retour en position centrale comme fonction « **SimDeselectOverride** » (annuler le mode DOGFIGHT ou MSL OVRD). Vous aurez besoin d'ajouter la ligne suivante dans le **falconbms.cfg**, pour que le mode DOGFIGHT ou MSL OVRD soit désactivé automatiquement si le bouton repasse en position centrale :

set g_bHotasDgftSelfCancel 1

falcon bms.cfg

```
////////////////////
// Misc Settings //
////////////////////

set g_bVoiceCom 1                              // Enable Voice Communications
set g_nF1TeamUiFreq 307300                     // Frequency for F1 UI voice comms (for Force on Force type missions).
set g_nF2TeamUiFreq 1234                       // Frequency for F2 UI voice comms (for Force on Force type missions).
set g_bhudAOA 1                                // Turns OFF the HUD AOA.
set g_bLocalEnvironmentalDate 0                // Makes the TimeManager use the local system date for environmental lighting calculations
set g_bHotasDgftSelfCancel 0                   // set to 1 = SRM and MRM override callbacks will call the override cancel callback for you
set g_nHotasPinkyShiftMagnitude 256            // Enable DirectX Shifting and specify button offset
set g_nHotasShiftQuickPressTimeLimit 200
set g_fFOVIncrement 5                          // Sets how much the field of view should change for each keypress in degrees
set g_fMaximumFOV 80                           // Limits the maximum amount that the FOV can be increased
set g_bNoAAAEventRecords 0                     // This option, set to 1, removes AAA shots from debrief
set g_bACMIRecordMsgOff 1                      // Turns off the ACMI RECORDING msg
set g_nPadlockBoxThickness 1                   // Determines the thickness of the padlock box
set g_nDynamicVoices 16                        // Maximum number of voices allocated by the sound code
set g_nSoundUpdateMS 10                        // How many milliseconds must elapse before the sound code updates
set g_nSoundSwitchFix 1                        // This patch may fix the problem of AI comms disappearing
set g_bPilotEntertainment 0                    // Enables the user to control WinAmp
set g_nWinAmpInitVolume 204                    // Sets the initial WimAmp volume
set g_fAmbientmin 0.1                          // Allows to tweak sky brightness at night
set g_bForce16bitDisplay 0                     // Forces 16 bit display device.
set g_bNewThreadTiming 1                       // If experiencing hiccups/stuttering in the UI with multi-core processors, try set to 0
```

Le chiffre **1** active la fonction. Chez moi, la fonction est désactivée car je n'ai pas ce problème lié au Cougar, donc je laisse le chiffre sur **0**.

Limites de la programmation DirectX :

Tout d'abord, il faut savoir que l'on ne peut pas programmer plus de 512 boutons DirectX, soit 16 contrôleurs de 32 boutons chacun. De plus, cette limite s'applique aussi bien aux fonctions non shiftées qu'aux fonctions shiftées. La somme des deux ne pourra excéder 512 boutons.

Ensuite, les numéros de boutons affichés dans BMS sont ceux reconnus par Windows. Il faut donc que le nombre de contrôleurs branchés lors de la configuration du fichier soit le même que lorsque vous allez jouer, sinon les numéros programmés ne correspondront plus avec le matériel branché, et vous aurez donc un décalage rendant totalement inutilisable le travail que vous avez effectué. Dans la mesure du possible, essayez de brancher chaque contrôleur sur le même port USB que la dernière fois, et dans le bon ordre. Ex : vous avez branché le stick sur le premier port USB, ensuite la throttle sur le deuxième port USB, puis en dernier le palo ou les MFD sur les troisièmes et quatrièmes ports, etc.... Normalement, les contrôleurs gardent les mêmes plages de numéros s'ils sont tous branchés. Chez moi, je me sers d'un volant reconnu comme un contrôleur pour mon palonnier. Les numéros de boutons du volant dans le fichier keystroke vont de 0 à 31. Les numéros de boutons de mon stick vont de 32 à 63, et ceux de ma throttle de 64 à 95. Si mes trois contrôleurs sont branchés, quelque soit le port utilisé, et l'ordre dans lequel je les branche, les plages de numéros seront correctes. Par contre, si mon volant n'est pas branché, alors les numéros de boutons de mon stick iront de 0 à 31, et ceux de ma throttle de 32 à 63. Le décalage rend donc le fichier keyfile inutilisable. Veillez donc à configurer votre profil avec les bons contrôleurs branchés.

Conclusion :

La programmation via un fichier keystroke est une très bonne solution de programmation pour un joystick. Elle permet d'éviter énormément de problèmes, et elle offre une possibilité de programmation conséquente avec un grand nombre de boutons, de Hats, et d'axes (programmables via le setup de BMS pour les axes). Les fonctions shiftées fonctionnent très bien, et elles permettent d'ajouter les fonctions utiles à tous ceux qui ne peuvent pas se construire un pit à la maison, et qui préfèrent utiliser le moins possible le clavier ! J'ai conçu un profil adapté à mes besoins pour mon x55 avec un fichier keystroke, et j'utilise mon clavier uniquement pour l'ICP, le menu de communication, et la commande d'éjection (le moins souvent possible pour cette dernière). Néanmoins, programmer en DirectX a aussi ses limites, et l'on ne peut pas accéder à une programmation avancée, comme ajouter des délais, effectuer une action lors du relâchement du bouton, etc... C'est pourquoi j'ai décidé de faire cohabiter deux profils. Le premier, c'est mon fichier keystroke qui programme 80% de mon HOTAS. Le deuxième profil, je l'ai créé avec mon soft de programmation Saitek, que j'ai utilisé pour programmer mon slider attribué aux aérofreins, et les sept interrupteurs à deux positions momentanées de la base du throttle. C'est une solution mixte qui me convient parfaitement. Je vous conseille donc de regarder du côté de la programmation

DirectX si vous venez de la découvrir, car bien maîtrisée, elle vous offre une solution de programmation très sûre. Si vous voulez programmer des fonctions plus avancées, il faudra vous tourner vers un logiciel complémentaire. Les logiciels de programmation fournis avec votre joystick font en général très bien l'affaire. N'oubliez pas de sauvegarder vos profils une fois fini, bien au chaud sur votre disque dur, car en cas de mauvaise manip', il est agréable de retrouver sa configuration en un simple copier-coller !

J'ai rédigé ce tuto sans prétention, et je me rends compte qu'il est très long. J'ai voulu le rendre le plus complet possible, en étant le plus précis possible sur ce qui pourrait poser problème. Je suis parti du principe que mon lecteur aurait besoin d'être guidé de A à Z, et j'ai conscience que tout le monde n'a pas besoin de ce niveau de précision. Ma volonté a donc rendu le tuto beaucoup moins synthétique. J'espère que ça ne le rendra pas pour autant moins accessible.

Je me suis appuyé en grande partie sur cet article présent sur le forum benchmark sims : http://www.benchmarksims.org/forum/cont ... g-Facility C'est avec cet article que j'ai appris à utiliser les fonctions DirectX. J'ai donc utilisé en partie le plan proposé par l'auteur, et il se peut que certaines parties de mon « article » présentent des ressemblances avec l'article en anglais. Sachez aussi qu'il est beaucoup plus court que mon tuto, si vous maîtrisez l'anglais, ça vaut peut-être le coup d'y faire un tour !

Le tuto est bien entendu ouvert aux critiques constructives ! Je suis preneur de vos avis, de vos remarques pour pouvoir l'améliorer afin de le rendre plus clair. Si quelqu'un a testé le keyfile generator de Kolbe, un petit paragraphe pourrait être également enrichir le tuto !

Merci de m'avoir lu jusqu'au bout pour ceux qui ont eu le courage et à bientôt !

Historique des modifications :

19/10/2014 : Rajout d'un lien direct vers le post de Checkpoint dans la deuxième partie pour le DirectX Generator de Kolbe. Rajout d'une ligne dans la partie "Connaître son HOTAS" concernant les spécificités des logiciels Foxy et TARGET, suite à une remarque de Mav-jp.

Tuto réalisé par Icewind

Lien vers le post d'Icewind

http://www.checksix-forums.com/viewtopic.php?f=281&t=187446&p=1446747#p1446747