

SUPER 68

I) Introduction

1) Description générale

Le héros du jeu se déplace sur un plateau en deux dimensions, en quête de clefs. Son but est de toutes les collecter, mais de nombreux obstacles entravent sa quête : hormis les murs qui lui barrent la route et les pièges qui le guettent, des fantômes passe-murailles le poursuivent! Heureusement, il a trois vies, et il peut se remettre des coups qui lui sont portés en absorbant des fioles de vie qui sont disséminées sur le plateau.

2) Description détaillée

a) Description du plateau de jeu

Les cases du plateau sont représentées en mémoire par un tableau d'octets. La valeur de chaque octet indique ce qui se trouve sur une case donnée. La matrice initiale, de 20 lignes par 80 colonnes (rangées par lignes), est donnée.

Les informations contenues dans une case (= un octet) sont organisées de la façon suivante :

b7	b6	b5	b4	b3	b2	b1	b0
M	P	V	C	B	F	F	H

Mur : indique la présence d'un mur sur cette case (si le bit est à 1),

Piège : indique la présence d'un piège

Vie : indique la présence d'une fiole de vie

Clef : indique la présence d'une clef,

Fantôme : champ réservé à la description d'un fantôme

00 : pas de fantôme dans cette case

01 : un fantôme est présent, sa prochaine action sera un déplacement

10 : un fantôme est présent, sa prochaine action sera une attente

11 : un fantôme est présent, ses deux prochaines actions seront des attentes

Héros : indique la présence du héros

Bonus : indique un bonus choisi par le concepteur

Remarque :

On ne peut pas avoir de fiole, de piège, ni de clef dans les murs. On ne peut pas non plus avoir de fiole, de piège, ou de clef confondus sur une même case.

b) Déplacement du héros

Le héros se déplace sur le plateau décrit précédemment. Il ne peut pas sauter de cases, ni traverser de mur. Il peut se déplacer uniquement dans quatre directions (haut, bas, gauche, droite), et d'une seule case par tour. Ses déplacements sont initiés par le joueur en pressant les touches du clavier par test d'état (le choix des touches à utiliser est laissé à l'appréciation des concepteurs).

c) Les clefs

La quête du héros consiste à ramasser toutes les clefs disséminées sur le plateau. Pour prendre une clef, le héros doit se trouver sur la même case que celle-ci; une fois cette clef prise, elle disparaît du

monde. L'ordre dans lequel cette collecte est effectuée n'a pas d'importance. Si toutes les clefs sont ramassées, le jeu se termine par la victoire du héros.

d) La vie

Au début de la partie, le héros possède trois vies.

Il perd des vies de deux manières :

- Premièrement, si par mégarde il marche sur un piège, il perd un point de vie.

Le piège ne disparaît pas.

- Deuxièmement, si le héros se trouve sur la même case qu'un fantôme, celui-ci l'attaque et lui fait perdre un point de vie. Les fantômes étant déjà morts, il ne peuvent être tués ni éliminés. Par conséquent, après la rencontre avec notre héros le fantôme reste présent sur sa case.

Heureusement, le héros peut regagner un point de vie en allant sur case contenant une fiole. Bien entendu, une fois bu (ce qui se fait automatiquement lorsqu'on se trouve sur la case), la fiole disparaît. Il faut cependant remarquer que le héros ne peut avoir que trois vies au plus.

Afin de rendre le nombre de vies visible pour l'utilisateur, nous utiliserons les diodes IA2 et IB2. Si le héros possède ses trois vies alors les deux diodes sont allumées, s'il en possède encore deux alors une seule diode est allumée et s'il ne lui reste plus qu'une vie alors les deux diodes sont éteintes.

e) Le temps

Le temps presse! Le héros doit récupérer toutes les clefs avant que le temps de jeu imparti ne soit écoulé. La visualisation du temps restant se fait grâce aux huit diodes : elles sont toutes allumées au début de la partie, et s'éteignent progressivement. Lorsqu'elles sont toutes éteintes, le temps est écoulé et le jeu se termine.

La gestion du temps dans le jeu est réalisée grâce aux timers, qui marquent le passage des tours de jeu. Le héros ne peut pas se déplacer plus d'une fois par tour; le plateau est mis à jour à chaque tour.

On doit avoir la possibilité de suspendre le jeu et de le reprendre à tout moment (gestion par interruption°. La pause (ou la sortie de pause) est déclenchée via le bouton poussoir IB1.

De plus, on a différents niveaux de jeu. Le niveau de jeu est lu sur les 8 interrupteurs à bascule au début de la partie. Plus le niveau est élevé, plus les tours de jeu sont rapides. Le nombre de niveaux et la façon dont ils sont codés sont laissés à l'initiative des concepteurs.

f) Les fantômes

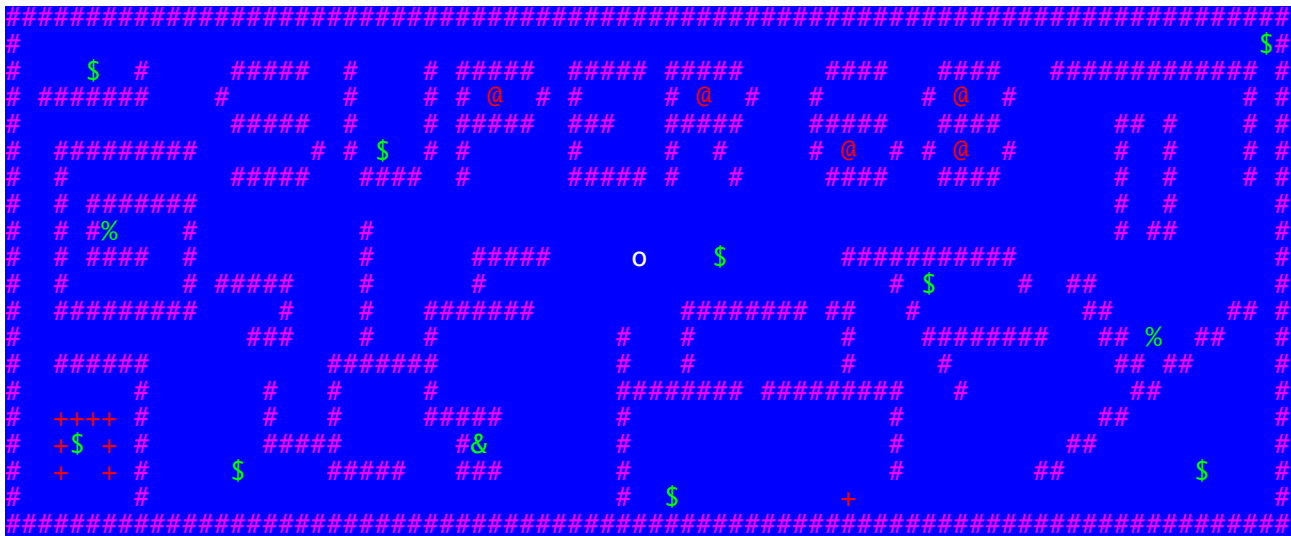
Le héros est confronté à plusieurs fantômes qui, en plus d'être immortels essaient volontairement de le tuer!

Les fantômes se déplacent de case en case comme le héros, mais ils sont plus lents : ils ne se déplacent ou n'effectuent d'action que tous les deux tours de jeu. N'étant pas faits de chair et de sang, ils ont la possibilité de passer à travers les murs (ainsi qu'à travers tous les obstacles : pièges, fioles, clefs). Quand un fantôme traverse un mur, il est ralenti : sa prochaine action sera une attente et ne pourra se déplacer qu'à l'action suivante. Il n'est pas ralenti par les autres obstacles.

Comme nous venons de le voir, les fantômes peuvent se trouver sur la même case que notre héros; dans ce cas, ils l'attaquent et lui prennent une vie. Une fois cette attaque effectuée, les fantômes doivent se reposer et perdent donc leurs deux actions suivantes, soient 4 tours de jeu.

Les fantômes se dirigent toujours, autant que possible, vers le héros. Il est important de noter que deux fantômes ne peuvent pas se trouver sur la même case.

II) Vue d'ensemble



: mur
+ : piège
% : vie
\$: clef
@ : fantôme
o : héros
& : sablier

III) Solutions retenues

- Le déplacement du héros s'effectue via les touches :
 - t : haut
 - g : bas
 - f : gauche
 - h : droite
- Le nombre de vies du héros est affiché en binaire à l'aide des deux diodes (plus logique)
 - 00 : 0 vie
 - 01 : 1 vie
 - 10 : 2 vies
 - 11 : 3 vies
- Les fantômes se déplacent vers le héros en comparant la distance les séparant sur l'axe des abscisses et des ordonnées; il se déplace en conséquence!
- Le niveau de difficulté lu sur les clefs a été implémenté comme ceci :
 - niveau 1 : 00000001
 - niveau 2 : 00000011
 - niveau 3 : 00000111
 - niveau 4 : 00001111
 - niveau 5 : 00011111
 - niveau 6 : 00111111

niveau 7 : 01111111

niveau 8 : 11111111

- Détails des niveaux de difficultés :

Niveaux	Temps/Diode	Temps Total	Temps d'1 tour
1	39 sec	5 min 12 sec	300 msec
2	36 sec	4 min 48 sec	270 msec
3	33 sec	4 min 24 sec	240 msec
4	30 sec	4 min 00 sec	210 msec
5	27 sec	3 min 36 sec	180 msec
6	24 sec	3 min 12 sec	150 msec
7	21 sec	3 min 48 sec	120 msec
8	18 sec	3 min 24 sec	090 msec

- Ajout de couleurs dans le jeu et ce qui l'entoure.

- Ajout d'un bonus : le sablier permettant au héros de gagner du temps.

- Ajout d'un menu simple dans le jeu :

```
Vous pouvez regler le niveau de difficile avec les clefs.  
niveau 1 : la clef de droite enclenchee.  
niveau 2 : les 2 clefs de droite enclenchees.  
etc.  
  
Les touches permettant de deplacer le heros sont :  
Haut : t      Gauche : f  
Bas  : g      Droite  : h  
  
Pour commencer a jouer, appuyez sur la touche espace.
```

- Un message est affiché quand le joueur a gagné ou perdu!

"Bravo! Vous avez gagne."

"Dommage! Vous avez perdu."

"Vous n'avez plus de temps!"

ou "Vous avez perdu toutes vos vies!"

IV) Les Fichiers

mvme162_base.s : Définition générale des ressources de la carte Motorola MVME162.

ip_base.s : Définition générale des ressources liées aux cartes IP-PIA.

inicard.s : Initialisation des clefs, des diodes et des timers.

- inicard

plateau.s : Le plateau de jeu.

string.s : Gère les E/S vers l'écran.

- putChar

- LF
- CR
- getChar
- putStr

conversion.s : Gère la conversion entre entier et chaînes de caractères.

- intToStr

opdisplay.s : Fonctions utiles en rapport avec l'affichage à l'écran.

- clrDisplay
- resetCursor
- setCursor
- setColor
- showCursor
- hideCursor

draw.s : Gère l'affichage des différents éléments du plateau de jeu.

- drawEmpty
- drawGhost
- drawHero
- drawWall
- drawTrap
- drawLife
- drawKey
- drawSand

display.s : Gère l'affichage du plateau de jeu.

- lDisplay
- gDisplay

keys.s : Gère ce qui concerne les clefs.

- updateKeys
- allKeys

life.s : Gère ce qui est en rapport avec les vies.

- updateLife
- updateTrap
- allLives

times.s : Gestion du temps.

- delTime
- cyclTime
- hourglass
- IT_Game
- IT_Cycle

button.s : Gestion des boutons.

- BN_Quit
- BN_Break

heromove.s : Gère le déplacement du héros.

- moveHero

- mvUp
- mvDown
- mvRight
- mvLeft

ghost.s : Gère le déplacement des fantômes.

- gGhost
- lGhost
- ghostHero

main.s : Le programme principal.

IV) Les Fonctions

1) inicard.s

a) inicard

But : Initialiser les deux timers, les diodes, les boutons et les clefs.

E/S : rien

Algo : Remettre à zéro les deux compteurs,
 définir les modes des compteurs,
 mettre à jour la table des interruptions,
 mettre les diodes en sortie,
 mettre les clefs en entrée, initialiser les boutons.

Utilisation :

bsr inicard	<pre> -4 +-----+ @ 0 +-----+ pile </pre>
------------------------	--

2) string.s

a) putChar

But : Afficher un caractère à l'écran.

E/S : Un caractère en entrée.

Algo : Récupérer le caractère,
 tester si le tampon est libre,
 mettre le caractère dans le buffer pour l'écrire.

Utilisation :

<pre> move.b #'A',-(A7) bsr putChar add.l #2,A7 </pre>	<pre> -10 +-----+ D0 -6 +-----+ @ -2 +-----+ e <- caractère 0 +-----+ pile </pre>
---	--

b) LF

But : Remplir la ligne courante de caractères espace ' '. (line feed)

E/S : Rien

Algo : Tester si le tampon est libre,
mettre dans le buffer le caractère "line feed" pour écriture.

Utilisation :

<pre> bsr LF </pre>	<pre> -4 +-----+ @ 0 +-----+ pile </pre>
--------------------------	--

c) CR

But : Aller au début de la ligne suivante. (carriage return)

E/S : Rien

Algo : Tester si le tampon est libre,
mettre dans le buffer le caractère "carriage return" pour écriture.

Utilisation :

<pre> bsr CR </pre>	<pre> -4 +-----+ @ 0 +-----+ pile </pre>
--------------------------	--

d) getChar

But : Récupérer un caractère tapé au clavier par l'utilisateur.

E/S : Le caractère lu en sortie.

Algo : Tester si un caractère a été reçu,
mettre le caractère dans la pile.

Utilisation :

suba.l #2,A7	-6 +-----+
bsr getChar	@
move.b (A7)+,D0	-2 +-----+
	s <- caractère
	+-----+
	pile

e) putStr

But : Afficher une chaîne de caractère à l'écran.

E/S : une chaîne de caractère en entrée(@).

Algo : Récupérer l'adresse de la chaîne de caractère,
tester si le caractère courant est le caractère de fin de chaîne,
empiler la valeur se trouvant à l'adresse courante,
incrémenter l'adresse (caractère suivant),
appeler la fonction putChar,
recommencer jusqu'au caractère de fin de chaîne.

Utilisation :

lea chaine,A0	-12 +-----+
move.l A0,-(A7)	A0
bsr putStr	-8 +-----+
add.l #4,A7	@
	-4 +-----+
	e <- adresse chaîne
	0 +-----+
	pile

3) conversion.s

a) intToStr

But : Convertir un nombre entier en chaîne de caractère.

E/S : un nombre en entrée.

Algo : Adressage auto-incrémenté dans 'number',
récupérer le nombre (quotient),
qui est dans la pile, empiler le caractère de fin de chaîne,
diviser le quotient par 10 et récupérer le reste,
convertir le reste en code ASCII,
mettre le reste dans la pile,
vérifier si le quotient est différent de zéro,
alors recommencer,
sinon récupérer les différents chiffre du nombre dans la pile et,
les mettre dans 'number' jusqu'au caractère de fin de chaîne.

Utilisation :

Algo : empiler 'ESC', appeler la fonction putChar, dépiler,
empiler '[', appeler la fonction putChar, dépiler,
récupérer le numéro de ligne,
appeler la fonction intToStr, dépiler,
empiler la chaîne de caractère (number),
appeler la fonction putStr, dépiler,
empiler ';', appeler la fonction putChar, dépiler,
récupérer le numéro de colonne,
appeler la fonction intToStr, dépiler,
empiler la chaîne de caractère (number),
appeler la fonction putStr, dépiler,
empiler 'H', appeler la fonction putChar, dépiler.

Utilisation :

<pre> move.l #36,-(A7) move.l #7,-(A7) bsr setCursor add.l #8,A7 </pre>	<pre> -16 +-----+ A0 -12 +-----+ @ -8 +-----+ e <- no ligne -4 +-----+ e <- no colonne 0 +-----+ pile </pre>
---	--

d) setColor

But : Changer la couleur de fond et de texte courante.

E/S : le numéro de la couleur de texte et de fond en entrée.

Algo : empiler 'ESC', appeler la fonction putChar, dépiler,
empiler '[', appeler la fonction putChar, dépiler,
récupérer le code de la couleur de texte,
appeler la fonction intToStr, dépiler,
empiler la chaîne de caractère (number),
appeler la fonction putStr, dépiler,
empiler ';', appeler la fonction putChar, dépiler,
récupérer le code la couleur de fond,
appeler la fonction intToStr, dépiler,
empiler la chaîne de caractère (number),
appeler la fonction putStr, dépiler,
empiler 'm', appeler la fonction putChar, dépiler.

Utilisation :

<pre> move.l #34,-(A7) move.l #47,-(A7) bsr setColor add.l #8,A7 </pre>	<pre> -16 +-----+ A0 -12 +-----+ @ -8 +-----+ e <- code texte -4 +-----+ e <- code fond 0 +-----+ pile </pre>
---	---

e) showCursor

But : Afficher le curseur à l'écran.

E/S : Rien

Algo : empiler 'ESC', appeler la fonction putChar, dépiler,
empiler '[', appeler la fonction putChar, dépiler,
empiler '?', appeler la fonction putChar, dépiler,
empiler '2', appeler la fonction putChar, dépiler,
empiler '5', appeler la fonction putChar, dépiler,
empiler 'h', appeler la fonction putChar, dépiler.

Utilisation :

bsr showCursor	-4 +-----+ @ 0 +-----+ pile
---------------------------	--

f) hideCursor

But : Masquer le curseur de l'écran.

E/S : Rien

Algo : empiler 'ESC', appeler la fonction putChar, dépiler,
empiler '[', appeler la fonction putChar, dépiler,
empiler '?', appeler la fonction putChar, dépiler,
empiler '2', appeler la fonction putChar, dépiler,
empiler '5', appeler la fonction putChar, dépiler,
empiler 'l', appeler la fonction putChar, dépiler.

Utilisation :

bsr hideCursor	-4 +-----+ @ 0 +-----+ pile
---------------------------	--

5) draw.s

a) drawEmpty

But : Afficher une case vide à l'écran.

E/S : Rien.

Algo : Changer les couleurs courantes (texte & fond) en appelant la fonction setColor.
Afficher le caractère correspondant : ' ' en appelant la fonction putChar.

Utilisation :

bsr drawEmpty	<pre> -4 +-----+ @ 0 +-----+ pile </pre>
--------------------------	--

b) drawGhost

But : Afficher un fantôme à l'écran.

E/S : Rien.

Algo : Changer les couleurs courantes (texte & fond) en appelant la fonction setColor.
Afficher le caractère correspondant : '@' en appelant la fonction putchar.

Utilisation :

bsr drawGhost	<pre> -4 +-----+ @ 0 +-----+ pile </pre>
--------------------------	--

c) drawHero

But : Afficher le héros à l'écran.

E/S : Rien.

Algo : Changer les couleurs courantes (texte & fond) en appelant la fonction setColor.
Afficher le caractère correspondant : 'o' en appelant la fonction putchar.

Utilisation :

bsr drawHero	<pre> -4 +-----+ @ 0 +-----+ pile </pre>
-------------------------	--

d) drawWall

But : Afficher un mur à l'écran.

E/S : Rien.

Algo : Changer les couleurs courantes (texte & fond) en appelant la fonction setColor.
Afficher le caractère correspondant : '#' en appelant la fonction putchar.

Utilisation :

bsr drawWall	<pre> -4 +-----+ @ 0 +-----+ pile </pre>
-------------------------	--

e) drawTrap

But : Afficher un piège à l'écran.

E/S : Rien.

Algo : Changer les couleurs courantes (texte & fond) en appelant la fonction setColor.

Afficher le caractère correspondant : '+' en appelant la fonction putchar.

Utilisation :

bsr	drawTrap	-4 +-----+ @ 0 +-----+ pile
-----	----------	--

f) drawLife

But : Afficher une fiole de vie à l'écran.

E/S : Rien.

Algo : Changer les couleurs courantes (texte & fond) en appelant la fonction setColor.

Afficher le caractère correspondant : '%' en appelant la fonction putchar.

Utilisation :

bsr	drawLife	-4 +-----+ @ 0 +-----+ pile
-----	----------	--

g) drawKey

But : Afficher une clef à l'écran.

E/S : Rien.

Algo : Changer les couleurs courantes (texte & fond) en appelant la fonction setColor.

Afficher le caractère correspondant : '\$' en appelant la fonction putchar.

Utilisation :

bsr	drawEmpty	-4 +-----+ @ 0 +-----+ pile
-----	-----------	--

h) drawSand

But : Afficher un sablier à l'écran.

E/S : Rien.

Algo : Changer les couleurs courantes (texte & fond) en appelant la fonction setColor.
Afficher le caractère correspondant : '&' en appelant la fonction putChar.

Utilisation :

bsr drawSand	<pre> -4 +-----+ @ 0 +-----+ pile </pre>
-------------------------	---

6) display.s

a) lDisplay

But : Afficher une case du plateau de jeu à l'écran.

E/S : La valeur d'une case du jeu en entrée.

Algo : Récupérer la valeur de la case qui se trouve dans la pile,
comparer cette case pour savoir si c'est une case vide, un fantôme, le héros,
un mur, un piège, une fiole de vie, une clef ou un sablier et,
appeler la fonction correspondante à la valeur de la case (drawEmpty, drawGhost,
drawHero, drawWall, drawTrap, drawLife, drawKey, drawSand)

Utilisation :

<pre> move.b (A0)+, -(A7) bsr lDisplay add.l #2, A7 </pre>	<pre> -10 +-----+ D0 -6 +-----+ @ -2 +-----+ e <- case du jeu 0 +-----+ pile </pre>
---	---

b) gDisplay

But : Afficher le plateau de jeu en entier à l'écran.

E/S : l'adresse du plateau de jeu.

Algo : Récupérer l'adresse du plateau de jeu de la pile,
pour la ligne 0 à 19 faire,
pour la colonne 0 à 79 faire,
empiler la valeur de l'@ courante,
incrémenter l'@ courante (case suivante),
appeler la fonction lDisplay,
dépiler,
incrémenter le nombre de colonnes,
fin pour,
mettre à zéro le nombre de colonnes et,
incrémenter le nombre de lignes,
appeler les fonctions LF et CR,
fin pour.

Utilisation :

<pre>lea PLATEAU,A0 move.l A0,-(A7) bsr gDisplay add.l #4,A7</pre>	<pre>-20 +-----+ A0 -16 +-----+ D1 -12 +-----+ D0 -8 +-----+ @ -4 +-----+ e <- @ plateau jeu 0 +-----+ pile</pre>
---	--

7) key.s

a) updateKeys

But : Mettre à jour le nombre de clefs.

E/S : @ courante du plateau et données du héros en entrée.

Algo : Récupérer l'@ du plateau et des données du héros,
vérifier si dans la case courante il y a une clef,
enlever la clef du plateau de jeu,
incrémenter le nombre de clefs du héros.

Utilisation :

<pre>move.l A1,-(A7) move.l A0,-(A7) bsr updateKeys add.l #8,A7</pre>	<pre>-20 +-----+ A1 -16 +-----+ A0 -12 +-----+ @ -8 +-----+ e <- @ plateau jeu e <- @ data héros 0 +-----+ pile</pre>
---	---

b) allKeys

But : Vérifier si le héros a récupéré toutes les clefs.

E/S : booléen en sortie, adresse des données du héros et nombre de clefs total du plateau de jeu en entrée.

Algo : Récupérer le nombre de clefs totatimes.sl,
récupérer l'@ des données du héros,
mettre la valeur du booléen en sortie à zéro,
si nombre de clefs du héros est égal au nombre de clefs total alors,
mettre la valeur du booléen en sortie à un.

Utilisation :

<pre> suba.l #2,A7 move.l A1,-(A7) move.b #nbkeys,-(A7) bsr allKeys add.l #6,A7 move.b (A7)+,D1 </pre>	<pre> -20 +-----+ A1 -16 +-----+ D0 -12 +-----+ @ -8 +-----+ e <- nb total clefs -6 +-----+ e <- @ data héros -2 +-----+ s <- booléen (res) 0 +-----+ pile </pre>
---	--

8) life.s

a) updateLife

But : Mettre à jour le nombre de fioles de vies.

E/S : @ courante du plateau et données du héros en entrée.

Algo : Récupérer l'@ du plateau et des données du héros,
vérifier si dans la case courante il y a une fiole de vie,
vérifier si le héros a moins de 3 vies,
enlever la fiole de vie du plateau de jeu,
incrémenter le nombre de vies du héros.

Utilisation :

<pre> move.l A1,-(A7) move.l A0,-(A7) bsr updateLife add.l #8,A7 </pre>	<pre> -20 +-----+ A1 -16 +-----+ A0 -12 +-----+ @ -8 +-----+ e <- @ plateau jeu -4 +-----+ e <- @ data héros 0 +-----+ pile </pre>
---	--

b) updateTrap

But : Mettre à jour le nombre de pièges.

E/S : @ courante du plateau et données du héros en entrée.

Algo : Récupérer l'@ du plateau et des données du héros,
vérifier si dans la case courante il y a une fiole de vie,
décrémenter le nombre de vies du héros.

Utilisation :

	-20 +-----+	
	A1	
	-16 +-----+	
	A0	
	-12 +-----+	
	@	
move.l A1, -(A7)	-8 +-----+	
move.l A0, -(A7)	e	<- @ plateau jeu
bsr updateTrap	-4 +-----+	
add.l #8, A7	e	<- @ data héros
	0 +-----+	
	pile	

c) allLives

But : Vérifier si le héros a au moins une vie et met à jour les diodes.

E/S : booléen en sortie, adresse des données du héros en entrée.

Algo : Récupérer l'@ des données du héros,
mettre la valeur du booléen en sortie à zéro,
éteindre les deux diodes,
vérifier si le héros a trois vies,
allumer les deux diodes,
mettre le booléen à un,
vérifier si le héros a deux vies,
allumer que la diode de gauche,
mettre le booléen à un,
vérifier si le héros a une vie,
allumer que la diode de droite,
mettre le booléen à un.

Utilisation :

	-22 +-----+	
	A1	
	-18 +-----+	
	D1	
	-14 +-----+	
	D0	
	-10 +-----+	
	@	
suba.l #2, A7	-6 +-----+	
move.l A1, -(A7)	e	<- @ data héros
bsr allLives	-2 +-----+	
add.l #4, A7	s	<- booléen (res)
move.b (A7)+, D1	0 +-----+	
	pile	

9) times.s

a) delTime

But : Calculer le temps total/diode de la partie à partir du niveau de difficulté choisi par l'utilisateur.

E/S : @ du tableau des valeurs du temps total / diode en entrée.

Algo : Récupérer l'@ du tableau des valeurs du temps total,
 comparer la valeur des clefs (niveau de difficulté choisi),
 récupérer la valeur correspondante au niveau de difficulté dans le
 tableau des valeurs du temps total,
 calculer la valeur en microsec du temps/diode,
 initialise la valeur du compteur à atteindre.

Utilisation :

<pre>lea tottime,A2 move.l A2,-(A7) bsr delTime add.l #4,A7</pre>	<pre>-20 +-----+ A2 -16 +-----+ D1 -12 +-----+ D0 -8 +-----+ @ -4 +-----+ e <- @ tottime 0 +-----+ pile</pre>
---	--

b) cyclTime

But : Calculer le temps d'un tour de jeu à partir du niveau de difficulté choisi par l'utilisateur.

E/S : @ du tableau des valeurs du temps d'un cycle de jeu en entrée.

Algo : Récupérer l'@ du tableau des valeurs du temps d'un cycle de jeu,
 comparer la valeur des clefs (niveau de difficulté choisi),
 récupérer la valeur correspondante au niveau de difficulté dans le
 tableau des valeurs du temps d'un cycle de jeu,
 calculer la valeur en microsec d'un tour de jeu,
 initialise la valeur du compteur à atteindre.

Utilisation :

<pre>lea trtime,A2 move.l A2,-(A7) bsr cyclTime add.l #4,A7</pre>	<pre>-20 +-----+ A2 -16 +-----+ D1 -12 +-----+ D0 -8 +-----+ @ -4 +-----+ e <- @ trtime 0 +-----+ pile</pre>
---	---

c) hourglass

But : Mettre à jour le nombre de sabliers.

E/S : @ courante du plateau et données du héros en entrée.

Algo : Récupérer l'@ du plateau,
 vérifier si dans la case courante il y a un sablier,
 enlever le sablier du plateau de jeu,
 récupérer le temps qu'il reste,
 ajouter le temps correspondant (une diode),
 mettre à jour l'affichage des diodes,
 décrémenter le nombre de vies du héros.

Utilisation :

<pre> move.l A0,-(A7) bsr hourglass add.l #4,A7 </pre>	<pre> -20 +-----+ A0 -16 +-----+ D0 -12 +-----+ @ -8 +-----+ e <- @ plateau jeu 0 +-----+ pile </pre>
---	---

d) IT_Game

But : Gérer le temps total d'une partie.

E/S : Rien

Algo : Récupérer le temps qu'il reste,
 éteindre une diode,
 remettre le compteur à zéro.

Utilisation :

<pre> interruption </pre>	<pre> -8 +-----+ D0 -4 +-----+ @ 0 +-----+ Pile </pre>
---------------------------	---

e) IT_Cycle

But : Gérer le temps d'un cycle de jeu.

E/S : Rien

Algo : - Récupérer le caractère saisi par l'utilisateur,
 empiler l'adresse du plateau,
 empiler l'adresse de herodata,
 appeler moveHero,
 dépiler,
 - empiler l'adresse du plateau,
 empiler l'adresse de herodata,
 empiler l'adresse de dataghost,
 appeler moveHero,

- dépiler,
- réserver un octet,
- empiler l'adresse de hérodata,
- empiler le nombre de clefs total,
- appeler allKeys,
- dépiler,
- si le héro a toutes les clefs,
- effacer l'écran et afficher le message gagné à l'écran,
- empiler l'adresse de hérodata,
- empiler l'adresse de dataghost,
- appeler ghostHero,
- dépiler,
- réserver un octet,
- empiler l'adresse de hérodata,
- appeler allLives,
- dépiler,
- si le nombre de vies est égal à zéro,
- effacer l'écran et afficher le message perdu à l'écran,
- si toutes les diodes sont éteintes,
- effacer l'écran et afficher le message perdu à l'écran,
- remettre le compteur à zéro.

Utilisation :

interruption	<pre> -4 +-----+ @ 0 +-----+ Pile </pre>
--------------	--

10) buttons.s

a) BN_Quit

But : Quitter le jeu.

E/S : Rien

Algo : Quitter la boucle infinie en mettant à un D0.

Utilisation :

interruption	<pre> -4 +-----+ @ 0 +-----+ Pile </pre>
--------------	--

b) BN_Break

But : Mettre le jeu en pause.

E/S : Rien

Algo : vérifier si D0 est égal à 0 alors,
arrêter les timers,

mettre la valeur 2 dans D0,
sinon vérifier si D0 est égal à 2 alors,
tester si utilisateur a appuyé sur le bouton alors,
relancer les timers,
mettre la valeur 0 dans D0,
sinon boucler.

Utilisation :

interruption	<pre> -4 +-----+ @ 0 +-----+ Pile </pre>
--------------	--

11) heromove.s

a) moveHero

But : Déplacer le héros.

E/S : Touche que l'utilisateur a tapée au clavier, adresse du plateau de jeu et des données du héros en entrée.

Algo : Récupérer l'adresse du plateau de jeu,
récupérer l'adresse des données du héros,
récupérer le caractère tapé par l'utilisateur,
vérifier si l'utilisateur a appuyé sur une touche de direction,
dans ce cas appeler la fonction correspondante :
empiler plateau & herodata,
mvLeft | mvRight | mvUp | mvDown,
dépiler.

Utilisation :

<pre> move.b SUCC_BUF, -(A7) move.l A0, -(A7) bsr moveHero add.l #10, A7 </pre>	<pre> -26 +-----+ A1 -22 +-----+ A0 -18 +-----+ D0 -14 +-----+ @ -10 +-----+ A1 -6 +-----+ A0 -2 +-----+ e <- caractère 0 +-----+ Pile </pre>
---	--

b) mvUp

But : Déplacer le héros vers le haut.

E/S : L'adresse du plateau de jeu et les données du héros en entrée.

Algo : Récupérer l'adresse du héros,
 récupérer l'adresse des données du héros,
 mettre dans des registres les coordonnées du héros,
 aller à la case au-dessus du héros $(y-1)*80+x$ -correction[cf VI],
 tester si la case est un mur,
 dans ce cas ne pas avancer,
 sinon empiler @ plateau & herodata et,
 appeler les fonctions de mise à jour de clefs, pièges, vies et sabliers,
 (updateKeys, updateTrap, updateLife, hourglass)
 mettre à jour la valeur de l'adresse du plateau courante,
 positionner le curseur à cette nouvelle case (setCursor),
 actualiser la case courante (IDisplay),
 mettre à jour les données du héros (heroy),
 aller à l'adresse de la case de dessous,
 effacer le héros dans la valeur de cette adresse,
 positionner le curseur à cette nouvelle case (setCursor),
 actualiser la case courante (IDisplay),

Utilisation :

move.l	A0, -(A7)	-32	+-----+	
move.l	A1, -(A7)		A1	
bsr	mvUp	-28	+-----+	
add.l	#8, A7		A0	
		-24	+-----+	
			D2	
		-20	+-----+	
			D1	
		-16	+-----+	
			D0	
		-12	+-----+	
			@	
		-8	+-----+	
			e	<- @ data héros
		-4	+-----+	
			e	<- @ plateau jeu
		0	+-----+	
				File

c) mvDown

But : Déplacer le héros vers le bas.

E/S : L'adresse du plateau de jeu et les données du héros en entrée.

Algo : Récupérer l'adresse du héros,
 récupérer l'adresse des données du héros,
 mettre dans des registres les coordonnées du héros,
 aller à la case au-dessous du héros $(y+1)*80+x$ -correction[cf VI],
 tester si la case est un mur,
 dans ce cas ne pas avancer,
 sinon empiler @ plateau & herodata et,
 appeler les fonctions de mise à jour de clefs, pièges, vies et sabliers,
 (updateKeys, updateTrap, updateLife, hourglass)

mettre à jour la valeur de l'adresse du plateau courante,
 positionner le curseur à cette nouvelle case (setCursor),
 actualiser la case courante (IDisplay),
 mettre à jour les données du héros (heroy),
 aller à l'adresse de la case de dessus,
 effacer le héros dans la valeur de cette adresse,
 positionner le curseur à cette nouvelle case (setCursor),
 actualiser la case courante (IDisplay),

Utilisation :

move.l	A0,-(A7)	-32	+-----+	
move.l	A1,-(A7)		A1	
bsr	mvDown	-28	+-----+	
add.l	#8,A7		A0	
		-24	+-----+	
			D2	
		-20	+-----+	
			D1	
		-16	+-----+	
			D0	
		-12	+-----+	
			@	
		-8	+-----+	
			e	<- @ data héros
		-4	+-----+	
			e	<- @ plateau jeu
		0	+-----+	
				File

d) mvLeft

But : Déplacer le héros vers la gauche.

E/S : L'adresse du plateau de jeu et les données du héros en entrée.

Algo : Récupérer l'adresse du héros,
 récupérer l'adresse des données du héros,
 mettre dans des registres les coordonnées du héros,
 aller à la case de gauche du héros $y*80+(x-1)$ -correction[cf VI],
 tester si la case est un mur,
 dans ce cas ne pas avancer,
 sinon empiler @ plateau & herodata et,
 appeler les fonctions de mise à jour de clefs, pièges, vies et sabliers,
 (updateKeys, updateTrap, updateLife, hourglass)
 mettre à jour la valeur de l'adresse du plateau courante,
 positionner le curseur à cette nouvelle case (setCursor),
 actualiser la case courante (IDisplay),
 mettre à jour les données du héros (herox),
 aller à l'adresse de la case de droite,
 effacer le héros dans la valeur de cette adresse,
 positionner le curseur à cette nouvelle case (setCursor),
 actualiser la case courante (IDisplay),

Utilisation :

move.l	A0,-(A7)	-32	+-----+	
			A1	
move.l	A1,-(A7)	-28	+-----+	
			A0	
bsr	mvLeft	-24	+-----+	
add.l	#8,A7		D2	
		-20	+-----+	
			D1	
		-16	+-----+	
			D0	
		-12	+-----+	
			@	
		-8	+-----+	
			e	<- @ data héros
		-4	+-----+	
			e	<- @ plateau jeu
		0	+-----+	
				File

e) mvRight

But : Déplacer le héros vers la droite.

E/S : L'adresse du plateau de jeu et les données du héros en entrée.

Algo : Récupérer l'adresse du héros,
récupérer l'adresse des données du héros,
mettre dans des registres les coordonnées du héros,
aller à la case de droite du héros $y*80+(x+1)$ -correction[cf VI],
tester si la case est un mur,
dans ce cas ne pas avancer,
sinon empiler @ plateau & herodata et,
appeler les fonctions de mise à jour de clefs, pièges, vies et sabliers,
(updateKeys, updateTrap, updateLife, hourglass)
mettre à jour la valeur de l'adresse du plateau courante,
positionner le curseur à cette nouvelle case (setCursor),
actualiser la case courante (IDisplay),
mettre à jour les données du héros (herox),
aller à l'adresse de la case de gauche,
effacer le héros dans la valeur de cette adresse,
positionner le curseur à cette nouvelle case (setCursor),
actualiser la case courante (IDisplay),

Utilisation :

	-32 +-----+	
	A1	
	-28 +-----+	
	A0	
	-24 +-----+	
	D2	
move.l A0,-(A7)	-20 +-----+	
move.l A1,-(A7)	D1	
bsr mvRight	-16 +-----+	
add.l #8,A7	D0	
	-12 +-----+	
	@	
	-8 +-----+	
	e	<- @ data héros
	-4 +-----+	
	e	<- @ plateau jeu
	0 +-----+	
	File	

12) ghost.s

a) gGhost

But : Déplacer les 5 fantômes.

E/S : Adresse du plateau de jeu, des données du héros, des données des fantômes en entrée.

Algo : Récupérer l'adresse du plateau de jeu dans la pile,
 récupérer l'adresse des données du héros dans la pile,
 récupérer l'adresse des données des fantômes dans la pile,
 Pour numéro fantôme de 0 à 4 faire,
 empiler l'adresse du plateau de jeu,
 empiler l'adresse des données du héros,
 empiler la valeur de l'adresse courante (ghosty) de dataghost,
 incrémenter l'adresse des données des fantômes,
 empiler la valeur de l'adresse courante (ghostx) de dataghost,
 incrémenter l'adresse des données des fantômes,
 appeler la fonction local ghost (lGghost),
 dépiler & mettre à jour les données des fantômes à partir de la pile,
 incrémenter le nombre numéro fantôme
 Fin pour.

Utilisation :

	-32 +-----+	
	A2	
	-28 +-----+	
	A1	
	-24 +-----+	
	A0	
move.l A0,-(A7)	-20 +-----+	
move.l A1,-(A7)	D0	
move.l A2,-(A7)	-16 +-----+	
bsr gGhost	@	
add.l #12,A7	-12 +-----+	
	e	<- @ data ghost
	-8 +-----+	
	e	<- @ data héros
	-4 +-----+	
	e	<- @ plateau jeu
	0 +-----+	
	File	

b) IGhost

But : Déplacer un seul fantôme.

E/S : Adresse du plateau de jeu et des données du héros en entrée, ordonnée et abscisse d'un fantôme en entrée et sortie.

Algo simplifié :

Vérifier ce que doit faire le fantôme ce tour ci : attendre ou avancer,
calculer la distance sur l'axe des x entre le fantôme et le héros,
calculer la distance sur l'axe des y entre le fantôme et le héros,
comparer ces deux distances,
déplacer le fantôme suivant l'axe où la distance est la plus grande,
déterminer si le fantôme va à droite, à gauche, en haut ou en bas,
tester si un autre fantôme se trouve déjà sur cette future case,
si c'est le cas essayer de déplacer le fantôme sur l'autre axe,
tester si la future case est un mur,
dans ce cas, le fantôme perd deux tours au lieu de un seul,
tester si dans la future case il y a le héros,
dans ce cas, le fantôme perd deux tours au lieu de un seul,
sinon le fantôme perd un seul tour,
mettre à jour la valeur de l'adresse du plateau de la case future,
positionner le curseur sur cette nouvelle case (setCursor),
actualiser cette case (IDisplay),
mettre à jour la valeur de l'adresse du plateau de l'ancienne case,
positionner le curseur sur cette ancienne case (setCursor),
actualiser cette case (IDisplay),
mettre dans la pile (sortie) les coordonnées du fantôme.

Utilisation :

	-48 +-----+	
	A1	
	-44 +-----+	
	A0	
	-40 +-----+	
	D5	
	-36 +-----+	
	D4	
move.l A0,- (A7)	-32 +-----+	
move.l A1,- (A7)	D3	
move.b (A2)+,- (A7)	-28 +-----+	
move.b (A2)+,- (A7)	D2	
bsr lGhost	-24 +-----+	
sub.l #1,A2	D1	
move.b (A7)+, (A2)	-20 +-----+	
sub.l #1,A2	D0	
move.b (A7)+, (A2)	-16 +-----+	
add.l #8,A7	@	
	-12 +-----+	
	e	<- @ plateau jeu
	-8 +-----+	
	e	<- @ data héros
	-4 +-----+	
	e/s	<- Ghost y
	-2 +-----+	
	e/s	<- Ghost x
	0 +-----+	
	File	

c) ghostHero

But : Vérifier si un fantôme et le héros sont sur la même case.

E/S : Adresse des données du héros et des données des fantômes en entrée.

Algo : Récupérer l'adresse des données du héros dans la pile,
 récupérer l'adresse des données des fantômes dans la pile,
 Vérifier si le héros a au moins une vie,
 Pour numéro fantôme de 0 à 4 faire,
 Comparer l'ordonnée du héros et du fantôme,
 Comparer l'abscisse du héros et du fantôme,
 Enlever une vie au héros,
 Fin pour.

Utilisation :

	-24 +-----+	
	A2	
	-20 +-----+	
	A1	
	-16 +-----+	
	D0	
	-12 +-----+	
	@	
	-8 +-----+	
	e	<- @ data ghost
	-4 +-----+	
	e	<- @ data héros
	0 +-----+	
	File	

13) main.s

But : Gérer le "programme".

- inclure les fichiers ip_base.s et mvme162_base.s,
- masquer les interruptions,
initialiser les timers, boutons, clefs et diodes,
permettre les interruptions,
- afficher le menu du jeu,
attendre que l'utilisateur appuie sur la touche espace,
- effacer l'écran,
cacher le curseur,
charger le plateau de jeu,
charger les données du héros,
afficher le plateau de jeu,
- initialiser le temps total de la partie,
initialiser le temps d'un tour de jeu,
- charger les données des fantômes,
allumer toutes les diodes,
démarrer les timers,
- rentrer dans la boucle infini,
- instructions de fin de programme,
- inclure les fichiers, string.s, conversion.s, inicard.s, button.s, plateau.s,
opdisplay.s, draw.s, display.s, key.s, heromove.s, times.s, life.s, menu.s,
ghost.s,
- variables de conversions,
- variables relatives au héros,
- variables de directions pour le héros,
- variables de couleurs du texte,
- variables de couleurs de fond,
- variables relatives au temps,
- variables relatives aux fantômes.

VI) Problèmes rencontrés

- Synchronisation entre plateau dont la première case est à la coordonnée (0,0) et le curseur dont la première case est à la coordonnée (1,1)
- Les fantômes : pas de problème réel, leur déplacement étant basé sur le même système que le héros.

VII) Améliorations possibles

- Accélérer l'affichage du plateau en comparant la case précédente pour éviter de changer les couleurs si c'est le même élément que l'on veut afficher. (Ceci implique des changements dans la fonction gDisplay et dans les fonctions draw qui doivent avoir en plus un booléen en entrée pour savoir si on doit changer de couleur)

- Ajouter un système de menu (Jouer, Option, Quitter). Option peut comprendre la règle du jeu, les touches pour jouer, le nombre de vies, le plateau de jeu, le nombre de fantômes présents, ...
- D'autres plateaux de jeu ce qui impliquent l'ajout des fonctions tels que la recherche des coordonnées du héros dans jeu ainsi que celles des fantômes.
- Pour pouvoir recommencer une partie il faudrait garder en mémoire avant de commencer à jouer les coordonnées du héros, des fantômes, des clefs, des vies et des sabliers présents sur le plateau de jeu.
- L'affichage en bas de l'écran (les 4 lignes qui restent) des touches du jeu, du nombre de vies du héros, du nombre de clefs ramassées, du temps exact qui lui reste.
- Gestion du score qui est plutôt difficile parce qu'il faut être juste par exemple par rapport au temps qu'il a mis pour ramasser toutes les clefs en fonction du niveau de difficulté choisi!

ESTIENNE Sébastien
2004