Practical 7
Space Invaders (Part 10)

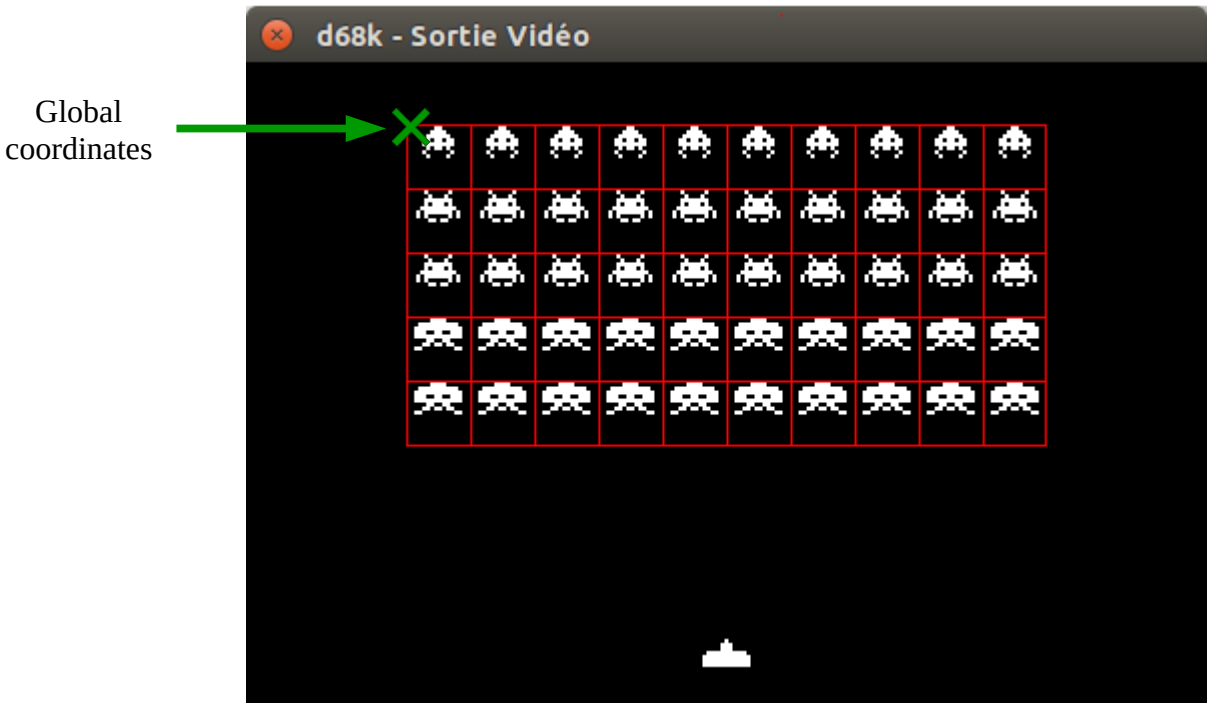
Step 1

We will start managing the invaders' movements. The invaders will move horizontally until they reach an edge of the screen. Then, they will go down one notch and go back horizontally in the other direction.

First, we need to define some constants that will be useful for the development of our subroutines:

Constant	Description
`INVADER_STEP_X`	Step increment in pixels for the horizontal displacement of the invaders.
`INVADER_STEP_Y`	Step increment in pixels for the vertical displacement of the invaders.
`INVADER_X_MIN`	Smallest global abscissa of the invaders. If the global abscissa of the invaders goes under this limit, it means that the invaders have reached the left edge.
`INVADER_X_MAX`	Largest global abscissa of the invaders. If the global abscissa of the invaders goes over this limit, it means that the invaders have reached the right edge.

As a reminder, the global position of the invaders is the coordinates of the top left corner of the very first square containing the invaders:



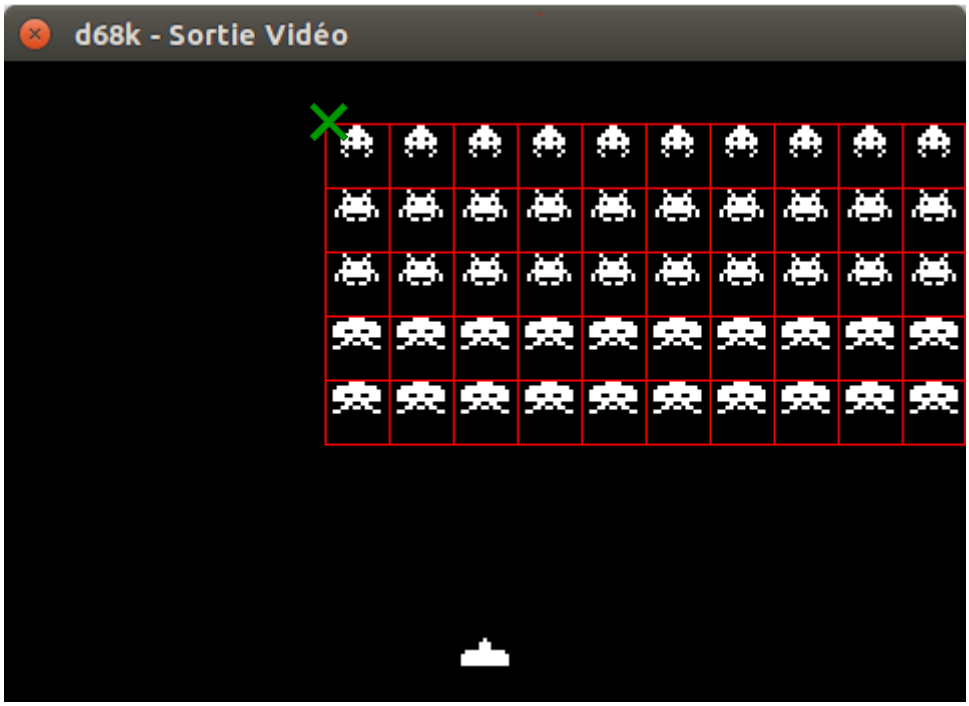
Add the following lines in the “Definitions of Constants” section of your source file:

```
INVADER_STEP_X      equ     4
INVADER_STEP_Y      equ     8
INVADER_X_MIN       equ     0
INVADER_X_MAX       equ     (VIDEO_WIDTH-(INVADER_PER_LINE*32))
```

According to the above numerical values, we can deduce that:

- When the invaders move to the left (with a 4-pixel step increment) and their global abscissa reaches 0, they have to move down by 8 pixels and go back to the right.

- When the invaders move to the right (with a 4-pixel step increment) and their global abscissa reaches 480 – 320 = 160, they have to move down by 8 pixels and go back to the left.



As it has been said previously, the global coordinates of the invaders are held in the `InvaderX` and `InvaderY` memory locations.

```
InvaderX            dc.w    (VIDEO_WIDTH-(INVADER_PER_LINE*32))/2       ; Global abscissa
InvaderY            dc.w    32                                          ; Global ordinate
```

We are going to add a memory location that will hold the horizontal step increment for the invaders in use.

```
InvaderX            dc.w    (VIDEO_WIDTH-(INVADER_PER_LINE*32))/2       ; Global abscissa
InvaderY            dc.w    32                                          ; Global ordinate
InvaderCurrentStep  dc.w    INVADER_STEP_X                              ; Current step
```

The `InvaderCurrentStep` memory location can only take two values:

- `INVADER_STEP_X` when the invaders move to the right.
- `-INVADER_STEP_X` when the invaders move to the left.

We will initialize it to `INVADER_STEP_X`, because the invaders start by moving to the right.

Now, write the **`GetInvaderStep`** subroutine that returns the next relative displacements of the invaders and updates the global coordinates as well as the current step increment.

<u>Outputs</u> : **D1.W** = Next horizontal relative displacement in pixels.
D2.W = Next vertical relative displacement in pixels.
`InvaderX`, `InvaderY` and `InvaderCurrentStep` are updated.

Tips:

- The subroutine must start by calculating the new global abscissa by adding the current step increment to the current global abscissa.
- If the new global abscissa is lower than `INVADER_X_MIN`, it is necessary to change direction.
- If the new global abscissa is greater than `INVADER_X_MAX`, it is necessary to change direction.
- If there is no change in direction:
  - ➔ The current step is loaded into **D1.W** (horizontal displacement according to the direction).
  - ➔ A step of 0 is loaded into **D2.W** (no vertical displacement).
  - ➔ `InvaderX` is updated with the new global abscissa.
- If there is a change in direction:
  - ➔ A step of 0 is loaded into **D1.W** (no vertical displacement).
  - ➔ A step of `INVADER_STEP_Y` is loaded into **D2.W** (vertical displacement).
  - ➔ `InvaderY` is updated (`INVADER_STEP_Y` is added).
  - ➔ The sign of `InvaderCurrentStep` is inverted.

Step 2

Write the **`MoveAllInvaders`** subroutine that moves all the invaders.

Tips:

- Start by determining the relative displacements by calling **`GetInvaderStep`**.
- Then apply these displacements (by calling **`MoveSprite`**) to all the invaders whose state is not `HIDE`.

Use the following main program in order to test your **`MoveAllInvaders`** subroutine (and at the same time, **`GetInvaderStep`**).

```
Main                jsr     InitInvaders

\loop               jsr     PrintShip
                    jsr     PrintShipShot
                    jsr     PrintInvaders
                    jsr     BufferToScreen

                    jsr     MoveShip
                    jsr     MoveAllInvaders
                    jsr     MoveShipShot

                    jsr     NewShipShot

                    bra     \loop
```

Check that the invaders move correctly from left to right, then from right to left. Also check that they go down with each change of direction. For the time being, stopping the invaders at the bottom of the screen is not taken into account. It will be managed soon in another step.

Step 3

For the moment, the ship moves at the same speed as the invaders. We will slow them down to make the ship faster. To do this, write the **`MoveInvaders`** subroutine that invokes the **`MoveAllInvaders`** subroutine every eighth call. That is to say:

- The first call to **`MoveInvaders`** calls **`MoveAllInvaders`**.
- The second call to **`MoveInvaders`** does nothing.
- The third call to **`MoveInvaders`** does nothing.
- Etc.
- The eighth call to **`MoveInvaders`** does nothing.
- The ninth call to **`MoveInvaders`** calls **`MoveAllInvaders`**.
- The tenth call to **`MoveInvaders`** does nothing.
- Etc.

Replace the "`jsr MoveAllInvaders`" by a "`jsr MoveInvaders`" in the previous main program and check that your ship is moving faster than the invaders.