Aspect environnemental :

1) Gestion de la mémoire (Valgrind) :

On utilise l'outil Valgrind afin de détecter les fuites mémoires du jeu et estimer leur importance.

On rappelle que les fuites mémoires peuvent résulter de plusieurs phénomènes :

- des blocs de mémoire alloués dont les adresses ne sont plus connues par l'application lorsque celle-ci se termine (Valgrind utilise le terme « Definitely Lost »)

- des blocs de mémoires dont des pointeurs contiennent leurs adresses et qui existent toujours à la fin de l'exécution du programme mais les pointeurs en question ne sont plus accessibles (Valgrind utilise le terme « Indirectly Lost »).

Il existe aussi d'autres phénomènes mais moins impactants.

On teste plusieurs parties avec plusieurs configurations (joueur vs ordinateur, ordinateur vs ordinateur, etc). Voici un rapport typique de Valgrind.

```
==19605== HEAP SUMMARY:
==19605==     in use at exit: 63,336 bytes in 2,549 blocks
==19605==   total heap usage: 3,840 allocs, 1,291 frees, 155,896 bytes allocated
==19605==
==19605== LEAK SUMMARY:
==19605==    definitely lost: 8,600 bytes in 501 blocks
==19605==    indirectly lost: 53,456 bytes in 2,016 blocks
==19605==      possibly lost: 0 bytes in 0 blocks
==19605==    still reachable: 1,280 bytes in 32 blocks
==19605==         suppressed: 0 bytes in 0 blocks
==19605== Rerun with --leak-check=full to see details of leaked memory
==19605==
==19605== For lists of detected and suppressed errors, rerun with: -s
==19605== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

On constate l'existence de fuites mémoires, qui en pratique peuvent représenter jusqu'à 50-60 ko sur le tas. C'est évidemment problématique, cependant la mémoire perdue est récupérée après chaque fin de partie, et compte tenu de l'ordre de grandeur des fuites mémoires, on peut raisonnablement affirmer que ces dernières ne pourront pas être à l'origine d'un arrêt brutal du jeu ou d'une saturation de la mémoire de la machine en question.

D'un autre côté, la mémoire perdue lors de l'exécution est tout de même conséquente si on la compare à la nature du jeu d'apparence « léger ». Ce point pourra donc être amélioré à l'avenir.

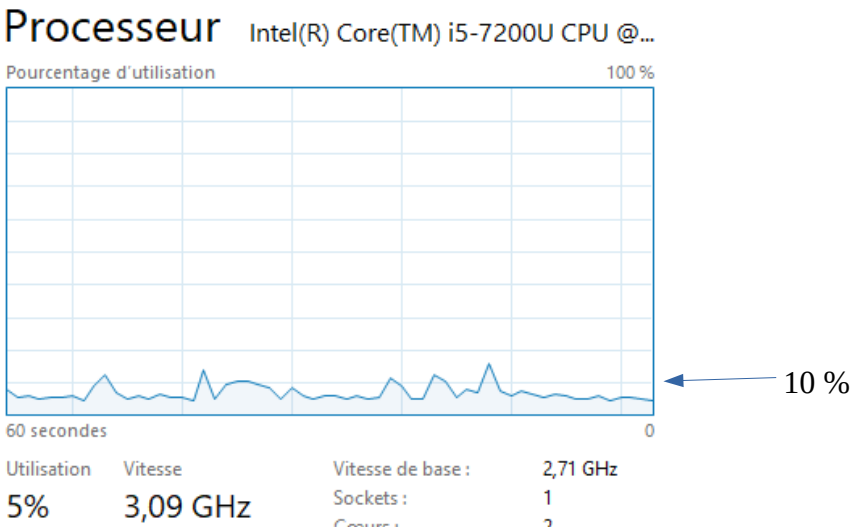
2) Temps CPU utilisé :

Le temps CPU est le temps exact que le CPU a passé à traiter des données pour un programme ou un processus spécifique.

Le temps CPU requis par les programmes et les processus est souvent minuscule, des fractions de seconde, c'est pourquoi de nombreux programmes peuvent être exécutés en même temps, mais continuent de tourner sur le CPU. Pour voir l'effet du jeu, on va regarder le taux d'utilisation du CPU (dans les mêmes conditions), et voir si celui ci varie significativement suivant que le programme est en exécution ou pas.

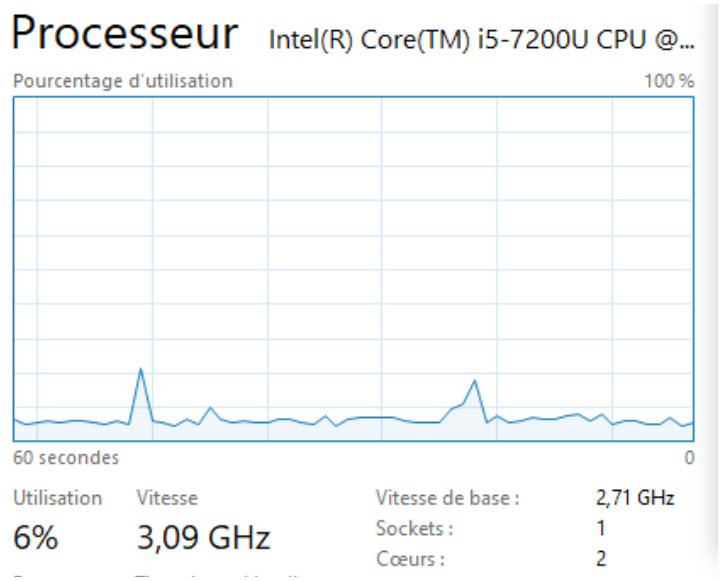
Programme non lancé :

Le taux d’utilisation du CPU tourne autour des 5-6 %.



Programme lancé :

On constate la même chose.



Sur ce point, le jeu ne semble avoir aucun impact, on peut aussi utiliser la commande time, si on prend l’exemple d’une partie joueur contre ordinateur, le temps CPU est d’à peine 0,1s par manche.

Finalement on peut conclure que le jeu développé à un faible impact en terme de ressources utilisées par la machine et donc par extension un faible impact environnemental.