

*Class Group Project*  
Development

Delivery

---

Group members

Documentation

**Conditions d'utilisations :** SUPINFO International University vous permet de partager ce document. Vous êtes libre de :

- Partager — reproduire, distribuer et communiquer ce document
- Remixer — modifier ce document

A condition de respecter les règles suivantes :

Indication obligatoire de la paternité — Vous devez obligatoirement préciser l'origine « SUPINFO » du document au début de celui-ci de la même manière qu'indiqué par SUPINFO International University – Notamment en laissant obligatoirement la première et la dernière page du document, mais pas d'une manière qui suggérerait que SUPINFO International University vous soutiennent ou approuvent votre utilisation du document, surtout si vous le modifiez. Dans ce dernier cas, il vous faudra obligatoirement supprimer le texte « SUPINFO Official Document » en tête de page et préciser notamment la page indiquant votre identité et les modifications principales apportées.

En dehors de ces dispositions, aucune autre modification de la première et de la dernière page du document n'est autorisée.

**NOTE IMPORTANTE :** Ce document est mis à disposition selon le contrat CC-BY-NC-SA Creative Commons disponible en ligne <http://creativecommons.org/licenses> ou par courrier postal à Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA modifié en ce sens que la première et la dernière page du document ne peuvent être supprimées en cas de reproduction, distribution, communication ou modification. Vous pouvez donc reproduire, remixer, arranger et adapter ce document à des fins non commerciales tant que vous respectez les règles de paternité et que les nouveaux documents sont protégés selon des termes identiques. Les autorisations au-delà du champ de cette licence peuvent être obtenues à [support@supinfo.com](mailto:support@supinfo.com).

© SUPINFO International University – EDUCINVEST - Rue Ducale, 29 - 1000 Brussels Belgium . [www.supinfo.com](http://www.supinfo.com)

# Table of contents





## TABLE DES MATIERES

1	RESUME DU GROUPE.....	4
2	RAPPORT DE PROJET .....	5
2.1	REPARTITION DES TACHES.....	5
3	MANUEL D'UTILISATION.....	6
3.1	INTRODUCTION.....	8
3.2	FONCTIONNEMENT DU JEU .....	9
3.3	CONFIGURATION REQUISE .....	10
3.4	INSTRUCTIONS D'INSTALLATION .....	10
3.5	COMMENT JOUER ?.....	10
3.6	SUPPORT TECHNIQUE .....	10
4	TECHNICAL DOCUMENTATION .....	11
4.1	MOTEUR DE JEU .....	11
4.1.1	Design général .....	11
4.1.2	Affichage des statistiques .....	11
4.1.3	Implémentation de dix niveaux.....	12
4.1.4	Passage au niveau suivant .....	13
4.1.5	Pause.....	13
4.2	GAMEPLAY.....	14
4.2.1	Mouvements du joueur .....	14
4.2.2	Destruction des blocs .....	14
4.2.3	Propagation de la destruction .....	15
4.2.4	Gestion du niveau d'oxygène .....	16
4.3	GESTION DU SCORE .....	17
4.3.1	Score calculé et affiché .....	17
4.3.2	Meilleurs scores.....	17
4.4	VICTOIRE / DEFAITE.....	18
4.4.1	Victoire après les 10 niveaux.....	18
4.4.2	Perte d'une vie.....	19
4.4.3	Défaite du joueur .....	20
4.5	FONCTIONNALITES BONUS .....	21
4.5.1	Animations.....	21
4.5.2	Musique .....	21

# 1 RESUME DU GROUPE

Campus : Strasbourg

Classe : **A.Sc.1**

ID Campus	Open	Nom de famille	Prénom	Photo
296188		GRUBER	Quentin	
295006		BIER	Maxime	
291960		CUSIMANO	Laurent	
280120		DUPREZ	Aurélie	

## 2 RAPPORT DE PROJET

---

### 2.1 REPARTITION DES TACHES

---

La répartition des tâches a été gérée principalement grâce au logiciel Git et son outil GitKraken, ainsi qu'avec l'application web Trello.

Trello est une application qui nous a permis de nous affecter des tâches du projet, ainsi que des dates limites pour être plus efficaces.

Ainsi, les tâches ont été globalement réparties de cette manière :

- Laurent a travaillé sur l'aspect visuel et sonore du jeu, en créant toutes les textures et les musiques, afin de rendre le jeu agréable à jouer et à regarder. Il a également travaillé sur l'affichage des niveaux, la mécanique de chute du joueur et l'interface graphique.
- Quentin a travaillé sur les animations des textures, ainsi que sur la génération des niveaux et des blocs qui les composent.
- Aurélie a travaillé sur tout ce qui concerne le score, sa progression et son stockage. Elle a aussi travaillé sur la gestion des menus principaux et pause.
- Maxime a travaillé sur le système d'oxygène et sur les propriétés des différents blocs. Il a aussi géré les interactions du joueur avec les blocs.

Bien entendu, les membres ne se sont pas arrêtés à ce qui leur a été affecté et ont aidé leurs collègues dans leurs parties.

### 3 MANUEL D'UTILISATION

---

## MR DRILLER

---



---

## TABLE DES MATIERES

---

3	MANUEL D'UTILISATION.....	6
3.1	INTRODUCTION.....	8
3.2	FONCTIONNEMENT DU JEU .....	9
3.3	CONFIGURATION REQUISE .....	10
3.4	INSTRUCTIONS D'INSTALLATION .....	10
3.5	COMMENT JOUER ? .....	10
3.6	SUPPORT TECHNIQUE .....	10

### 3.1 INTRODUCTION

---

Dans Mr. Driller, vous incarnez un jeune joueur qui, lors d'une de ses sessions habituelles à la salle d'arcade, s'effondre sur le sol, happé dans le jeu par une force mystérieuse.

Afin de retrouver sa vie réelle, il va devoir parcourir plusieurs lieux et percer à jour les multiples secrets enfouis dans les profondeurs de ce monde virtuel.

Mr. Driller est un jeu 2D dans lequel on incarne un personnage muni d'une foreuse. Le but du jeu est de creuser dans le sol le plus profondément possible afin d'atteindre la fin du niveau, tout en évitant de mourir.



## 3.2 FONCTIONNEMENT DU JEU

---

Le jeu possède un total de 10 niveaux, dont 2 niveaux spéciaux.

Le personnage a une jauge d'oxygène qui baisse au fur et à mesure du niveau. Il commence avec une jauge remplie à 100, et il perd 1 point d'oxygène par seconde. Le niveau d'oxygène peut être altéré par les différents blocs. Le joueur perd une vie lorsque son niveau d'oxygène atteint 0.

Le jeu contient un total de 5 types de blocs différents :

- Les blocs normaux (Violet, rose, turquoise, orange) : Ces blocs ont 1 point de vie et se détruisent lorsqu'on les perce. La destruction se propage aux blocs adjacents de la même couleur.
- Les blocs métalliques : Ces blocs ont 5 points de vie, il faut donc les percer 5 fois pour les détruire. Ils font perdre 20 points d'oxygène lors de leur destruction, et celle-ci ne se propage pas aux blocs adjacents.
- Les blocs blancs : Ces blocs ont les mêmes caractéristiques que les blocs normaux, mais leur destruction ne se propage pas aux autres blocs blancs.
- Les blocs de cristal : Ces blocs mettent quelques secondes à disparaître après les avoir tapés une première fois.
- Les capsules d'oxygène : Elles font regagner 20 points d'oxygène au joueur lorsqu'il marche dessus.

Le joueur est soumis à la gravité, il tombe lorsqu'aucun élément placé sous lui ne le retient.

Le joueur a 3 vies (2 résurrections). Lorsqu'il meurt une dernière fois, la partie est terminée et le joueur a perdu. À chaque résurrection, les blocs au-dessus du joueur sont détruits sans propagation.

Le score du joueur est calculé pendant chaque partie en fonction de la destruction des blocs, et les meilleurs scores sont répertoriés dans un classement local.

### 3.3 CONFIGURATION REQUISE

---

- Un clavier
- Un écran
- Une version de python 3 ou plus

### 3.4 INSTRUCTIONS D'INSTALLATION

---

- Télécharger les fichiers du jeu
- Lancer l'exécutable nommé MrDriller.exe

### 3.5 COMMENT JOUER ?

---

Le joueur se déplace avec les touches Q (gauche) et D (droite).

Il peut détruire les blocs adjacents avec les 3 flèches directionnelles [bas, gauche, droite]

Le jeu se met en pause avec la touche Échap, et les options du menu sont sélectionnées avec les flèches directionnelles et la touche Entrée.

### 3.6 SUPPORT TECHNIQUE

---

En cas de problème, n'hésitez pas à contacter le support technique à l'adresse suivante :

**291960@supinfo.com**

## 4 TECHNICAL DOCUMENTATION

---

### 4.1 MOTEUR DE JEU

---

#### 4.1.1 Design général

---

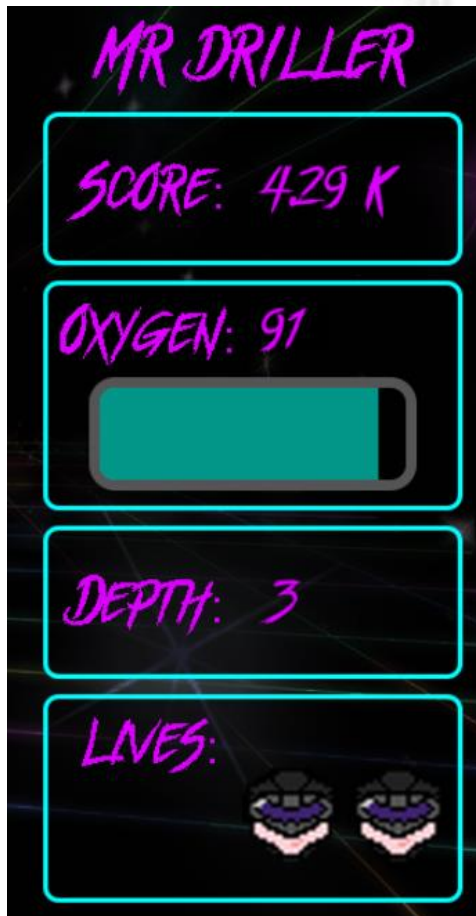
Le jeu est affiché dans une fenêtre pygame d'une taille de 800x600 pixels. Des textures ont été ajoutées pour chaque élément du jeu, et elles sont affichées unes par unes. Un fond différent a été ajouté pour chaque niveau.



#### 4.1.2 Affichage des statistiques

---

Les statistiques en jeu du joueur sont affichées dans un bandeau sur le côté de l'écran. On peut y lire le score actuel, le niveau d'oxygène accompagné d'une jauge représentative, sa profondeur dans le niveau actuel et son nombre de résurrections restantes :



```
if not inPause and not inMenu and not isDead and not won:
    surface.blit(Ui_bg, (0, 0))
    surface.blit(score_display, (640, 107))
    surface.blit(Oxygen_display, (640, 200))
    surface.blit(oxyImage, (537, 252))
    surface.blit(Depth_display, (640, 377))
```

Le code ci-dessus affiche les différents éléments sur la fenêtre de jeu en suivant les coordonnées. Il affiche d'abord le fond de la fenêtre, puis il y superpose le score, le niveau d'oxygène, la barre d'oxygène, la profondeur et le nombre de vies restantes.

#### 4.1.3 Implémentation de dix niveaux

Le jeu possède un total de 10 niveaux, dont 2 niveaux spéciaux. Les premiers niveaux ont une profondeur de 75 blocs, puis la profondeur passe à 150 blocs pour les niveaux plus difficiles.

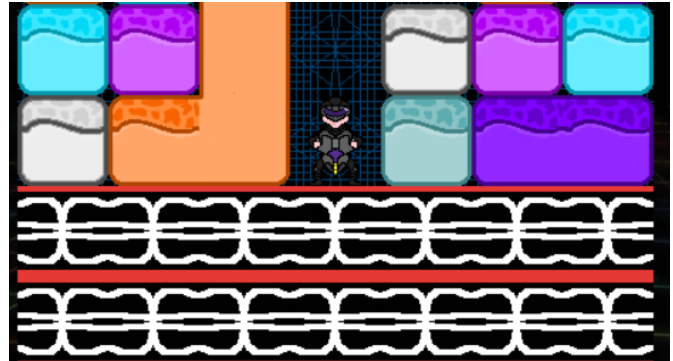
Les différents paramètres du niveau peuvent être facilement modifiés dans la fonction *generateLvl*, elle est composée d'un grand nombre de paramètres qui permettent de moduler la génération des niveaux. Il a donc été plus simple de générer des niveaux spéciaux.

```
def generateLvl(colors, lines, width, background, pillP=5, PillPL=7,
               PillMLE=20, SoloP=10, UnbreakableP=5, DelayedP=10):
```

#### 4.1.4 Passage au niveau suivant

Le joueur passe au niveau suivant dès qu'il atteint la fin du niveau actuel. La fin du niveau est détectée par des blocs spéciaux, qui lancent la fonction pour changer de niveau lorsqu'ils sont détruits.

```
elif self._blockType == "end":
    self.changeLvl()
    self._hp -= 1
```



#### 4.1.5 Pause

Le jeu peut être mis en pause à n'importe quel moment. L'écran de pause contient trois options : Resume pour reprendre la partie, Restart pour recommencer la partie et Quit pour revenir au menu.

Le programme détecte l'état du jeu en fonction des variables *inGame* et *inPause* qui valent *True* ou *False*. Lorsque le joueur appuie sur la touche échap, le jeu échange les valeurs de *inGame* et *inPause*, ce qui change l'état du jeu. Quand le jeu est en pause, les fonctions changent et le compteur d'oxygène s'arrête.



## 4.2 GAMEPLAY

### 4.2.1 Mouvements du joueur

Le joueur peut se déplacer à gauche et à droite grâce à l'appui des touches Q et D sur le clavier. Il peut également « grimper » sur les blocs adjacents, de la même manière que sur un escalier.

Le programme détecte l'appui des touches du clavier, et lance la fonction de déplacement avec le paramètre approprié pour le déplacement.

```
def mouvementHandle(event, surface, player, level, movKeys):  
  
    if event.key == movKeys[2]:  
        player.move(surface, 4, level)  
    elif event.key == movKeys[1]:  
        player.move(surface, 2, level)  
    elif event.key == movKeys[0]:  
        player.move(surface, 1, level)
```

Le programme vérifie si la case de destination est libre avant de déplacer le joueur. Dans le cas d'un déplacement diagonal, le programme vérifie également si les cases à travers lesquelles le joueur se déplace sont libres. Si c'est le cas, le joueur peut se déplacer.

### 4.2.2 Destruction des blocs

Le joueur peut détruire un bloc à gauche, à droite et en dessous de lui-même. Les commandes de destruction des blocs sont les flèches directionnelles. Le programme détecte l'appui de ces touches de la même manière que les mouvements.

```
def breaking(event, surface, player, level, currentBotLine):  
  
    if event.key == K_RIGHT:  
        player.breakBlock(surface, 2, level, currentBotLine)  
    elif event.key == K_DOWN:  
        player.breakBlock(surface, 3, level, currentBotLine)  
    elif event.key == K_LEFT:  
        player.breakBlock(surface, 4, level, currentBotLine)
```

La plupart des blocs ont 1 point de vie. L'action de percer un bloc fait perdre 1 point de vie aux blocs concernés. Lorsque les blocs ont 0 point de vie, leur hitbox disparaît et leur texture est remplacée par le fond du niveau. Le joueur peut alors se déplacer sur cette case.



Les blocs métalliques ont quant à eux 5 points de vie. Chaque forage fait également perdre 1 point de vie à ces blocs, et ils disparaissent également lorsque leurs points de vie sont à 0. L'avancement de la destruction de ces blocs est représenté par une fissure qui progresse à travers le bloc au fur et à mesure de celle-ci.



#### 4.2.3 Propagation de la destruction

Lorsque le joueur détruit un bloc basique, le programme vérifie autour de ce bloc s'il y a des blocs identiques, et si les conditions pour que la destruction sont respectées grâce au code suivant :

```
if self._chain_reaction == 1 and nochain == 0 and self._blockType == "classic":

    if level[self._posY + 1][self._posX].hpAccess() != 0 \
        and level[self._posY + 1][self._posX].typeAccess() == "classic":

        if level[self._posY + 1][self._posX].ColorAccess() == self._colors:
            level[self._posY + 1][self._posX].hit(surface, level, player)

    if level[self._posY - 1][self._posX].hpAccess() != 0 \
        and level[self._posY - 1][self._posX].typeAccess() == "classic":

        if level[self._posY - 1][self._posX].ColorAccess() == self._colors:
            level[self._posY - 1][self._posX].hit(surface, level, player)

    if self._posX < len(level[0]) - 1 and level[self._posY][self._posX + 1].hpAccess() != 0 \
        and level[self._posY][self._posX + 1].typeAccess() == "classic":

        if level[self._posY][self._posX + 1].ColorAccess() == self._colors:
            level[self._posY][self._posX + 1].hit(surface, level, player)

    if level[self._posY][self._posX - 1].hpAccess() != 0 and self._posX > 0 \
        and level[self._posY][self._posX - 1].typeAccess() == "classic":

        if level[self._posY][self._posX - 1].ColorAccess() == self._colors:
            level[self._posY][self._posX - 1].hit(surface, level, player)
```

Par exemple, la destruction en chaîne des blocs n'a pas lieu lors de la réapparition du joueur.

#### 4.2.4 Gestion du niveau d'oxygène

Tout au long de la partie, le joueur doit gérer son niveau d'oxygène afin de ne pas mourir. Il commence la partie avec 100 points d'oxygène, et il perd 1 point d'oxygène par seconde grâce à un événement de « timer ». Son niveau d'oxygène est représenté sur le côté de l'écran par une jauge et un compteur.

Le joueur peut interagir avec différents blocs du jeu pour modifier son niveau d'oxygène, tel que les capsules d'oxygène et les blocs métalliques.

Un bloc métallique enlève 20 points d'oxygène au joueur lorsqu'il est détruit, tandis qu'une capsule d'oxygène en ajoute 20. Leur destruction est détectée par le code suivant :

```
if self._blockType == "unbreakable":
    self._hp -= 1
    self.updTexture()
    if self.hpAccess() == 0:
        self._brkSound.play()
        player.updateOxygen(2, surface, level)
        player.AddScore(10)
    else:
        self._hitSound.play()

elif self._blockType == "pill":
    player.updateOxygen(3, surface, level)
    player.AddScore(20)
    self._brkSound.play()
    self._hp -= 1
```

Noter que les capsules sont ramassées lorsque le joueur marche dessus, et non quand il essaye de les percer.

La fonction pour mettre à jour l'oxygène est la même, et les différents cas sont triés dans la fonction elle-même. Le cas n°1 se déclenche toutes les secondes, le n°2 se déclenche lors de la destruction d'un bloc métallique et le n°3 se déclenche lorsque le joueur ramasse une capsule d'oxygène.

```
def updateOxygen(self, funct, surface, level):
    if funct == 1:
        self.__oxygen -= 1
    elif funct == 2:
        self.__oxygen -= 20
    elif funct == 3:
        if self.__oxygen <= 70:
            self.__oxygen += 20
        else:
            self.__oxygen = 100

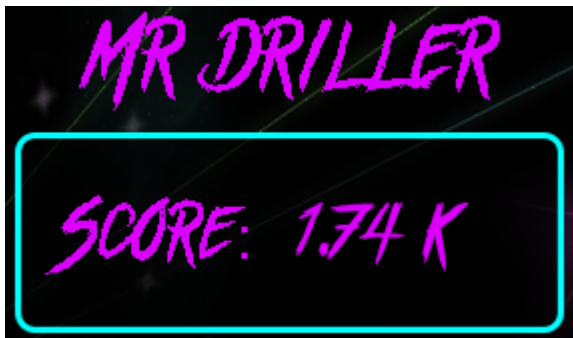
    if self.__oxygen <= 0:
        ugh = pygame.mixer.Sound(path.join("Assets", "Sounds", "ugh.wav"))
        ugh.set_volume(0.70)
        ugh.play(0)
        self.revive(surface, level)
```



## 4.3 GESTION DU SCORE

### 4.3.1 Score calculé et affiché

Le score du joueur augmente tout au long de la partie. Chaque bloc donne un certain nombre de points au joueur lorsqu'il est détruit.



```
if self.hpAccess() == 0:
    self._brkSound.play()
    player.updateOxygen(2, surface, level)
    player.AddScore(10)
```

Le score du joueur est affiché sur le côté de l'écran au cours de la partie.

### 4.3.2 Meilleurs scores

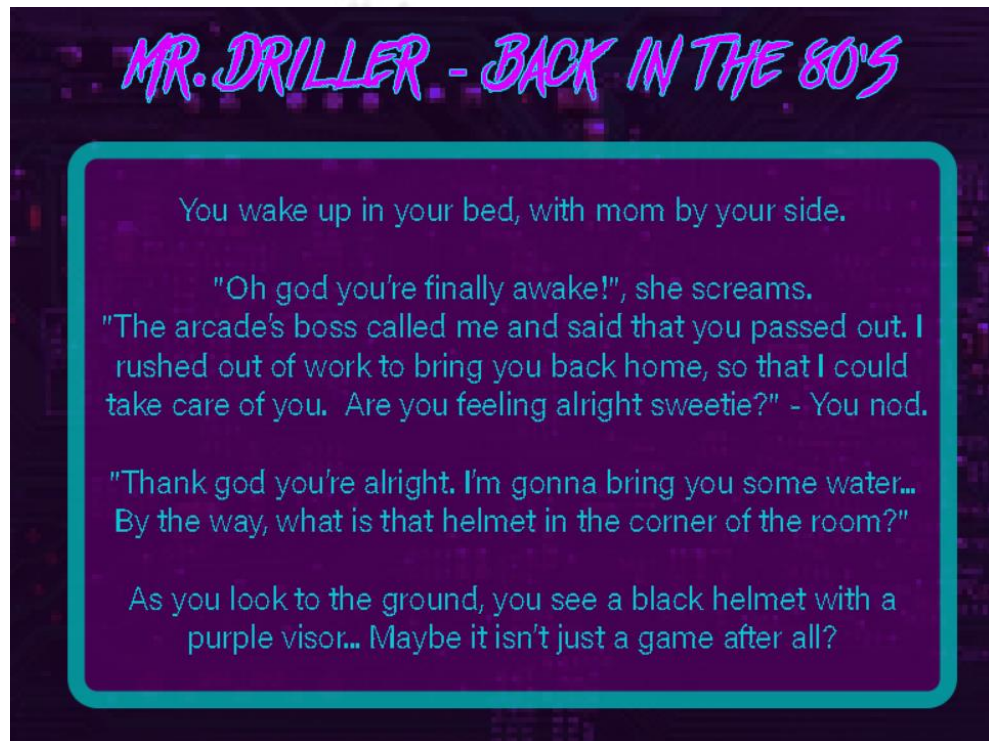
Après chaque victoire, le score du joueur au moment de sa victoire est stocké dans un fichier texte. Ce fichier texte contient l'historique des 3 meilleurs scores du joueur, et ces scores sont affichés sur l'écran d'accueil avant de lancer la partie.

1 : 26050 2 : 15164 3 : 13150

## 4.4 VICTOIRE / DEFAITE

### 4.4.1 Victoire après les 10 niveaux

Après avoir terminé le dixième niveau, le joueur a gagné la partie. Il voit alors l'écran suivant :

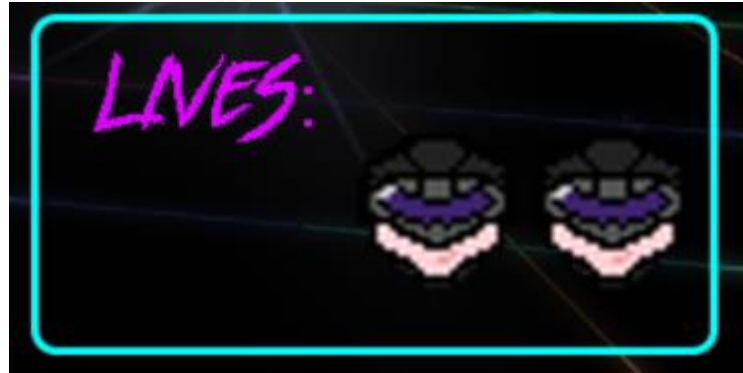


Le programme détecte la fin de chaque niveau, et cette partie de code se déclenche seulement à la fin du dixième niveau :

```
else:
    storeScore(player.scoreAcc())
    won = True
    lvl = [[]]
    return lvl, currentLvl, won
```

#### 4.4.2 Perte d'une vie

Dans le jeu, le personnage perd une vie si son niveau d'oxygène tombe à zéro. Dans le jeu, un compteur représentant le nombre de vies restant est affiché au premier plan :



Le programme détecte lorsque le compteur d'oxygène atteint 0. Si c'est le cas, il lance la procédure de résurrection.

```
if self.__oxygen <= 0:
    ugh = pygame.mixer.Sound(path.join("Assets", "Sounds", "ugh.wav"))
    ugh.set_volume(0.70)
    ugh.play(0)
    self.revive(surface, level)
```

```
def revive(self, surface, level):

    if self.__lives <= 0:
        self.__lives -= 1
        self.__oxygen = 100

    else:
        self.__IsReviving = True
        self.__oxygen = 100
        self.__lives -= 1

        for i in range(-2, 1):
            if (self.__posX + 1) != 7:
                if level[self.__posY + i][self.__posX + 1].hpAccess() != 0:
                    level[self.__posY + i][self.__posX + 1].hit(surface, level, self, 1, 1)
            if (self.__posX - 1) != -1:
                if level[self.__posY + i][self.__posX - 1].hpAccess() != 0:
                    level[self.__posY + i][self.__posX - 1].hit(surface, level, self, 1, 1)
            if level[self.__posY + i][self.__posX].hpAccess() != 0:
                level[self.__posY + i][self.__posX].hit(surface, level, self, 1, 1)

        self.__oxygen = 100
```

Cette fonction va analyser le nombre de vies restantes du joueur. S'il lui reste des vies, il va pouvoir lancer sa procédure de résurrection.

Cette procédure change l'aspect du personnage, et les blocs autour du personnage sont détruits. L'oxygène est également réinitialisé, et le joueur perd une vie.



#### 4.4.3 Défaite du joueur

La fonction qui détecte la mort du joueur est la même que celle utilisée ci-dessus.

Pendant le jeu, le programme vérifie régulièrement si le nombre de vies du joueur est inférieur à 0.

```
while inProgress:
    if player.livesAcc() < 0 and not isDead
        isDead = True
        inGame = False
```

Si c'est le cas, la procédure de jeu sort de l'état *inGame* et entre dans l'état *isDead*. Cette procédure va empêcher la progression dans le jeu et va afficher cet écran de mort :



Le joueur a alors la possibilité de recommencer la partie, ou de retourner vers le menu. Son score n'est pas enregistré.

## 4.5 FONCTIONNALITES BONUS

---

### 4.5.1 Animations

---

Le joueur a une animation correspondant à tous ses états. Ces animations se déclenchent à tour de rôle tout au long de la partie.

Les animations en cours sont gérées par différents états qui alternent entre *True* et *False* tout au long de la partie.

```
self.__IsMovingRight = False
self.__IsMovingLeft = False
self.__IsFalling = False
self.__IsDrillingRight = False
self.__IsDrillingLeft = False
self.__IsDrillingRight_off = False
self.__IsDrillingLeft_off = False
self.__IsDrillingDown = False
self.__IsReviving = False
self.__IsIdling = True
```

Le résultat en jeu est agréable à regarder.

### 4.5.2 Musique

---

Le jeu contient des bruitages et de la musique qui varie en fonction de tous les niveaux. Ces musiques ont été composées par nous-mêmes, et donnent une ambiance spéciale à chaque niveau.

-  Level1.wav
-  Level2.wav
-  Level3.wav
-  Level4.wav
-  Level5.wav
-  Level6.wav
-  Level7.wav
-  Level8.wav
-  Level9.wav
-  Level10.wav
-  menu.wav
-  win.wav