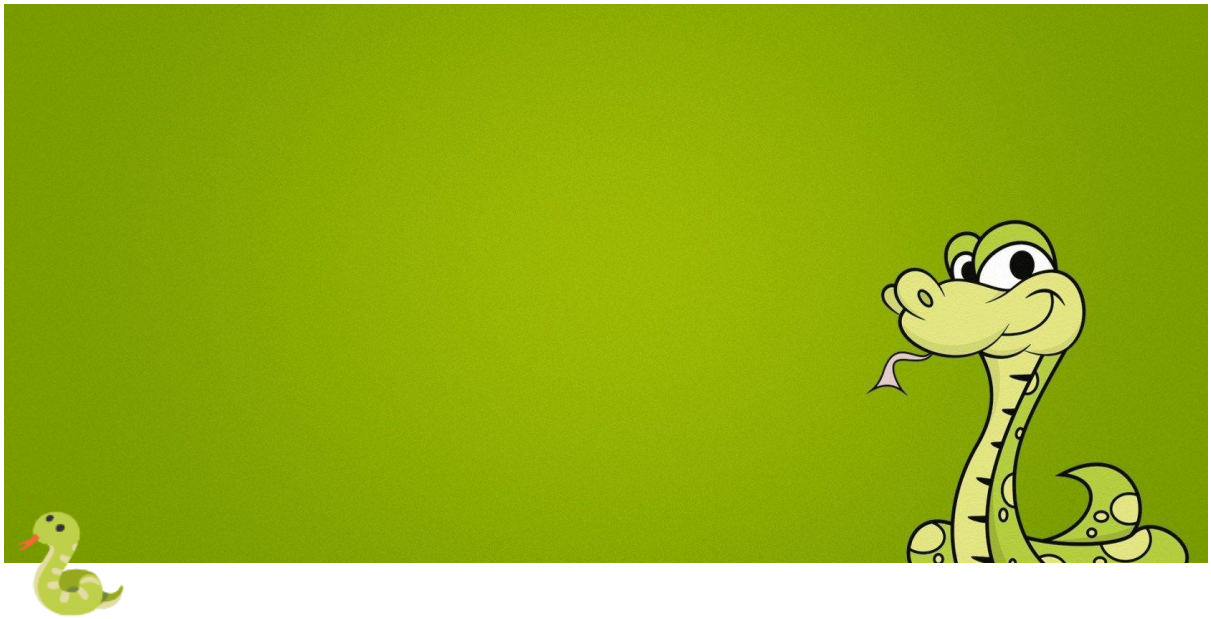


Snake JS de A à Z

"SSSsssSSS'est que je commenSSSsssSSSe à avoir faim moi" se dit Snaky, votre serpent de compagnie favoris. Et oui, un serpent aussi ça a besoin de manger ! Mais comme son maitre n'est pas à la maison, il décide d'aller chercher sa nourriture lui même.

Parviendrez-vous à l'aider dans cette tâche ?

Consignes

- Si vous avez des questions, pensez à demander de l'aide à votre voisin de droite. Puis de gauche. Demandez enfin à un Cobra (ceux-là ne mordent pas) si vous êtes toujours bloqué(e).
- Vous avez tout à fait le droit d'utiliser internet pour trouver des réponses ou pour vous renseigner.
- N'hésitez pas à faire des bonus et à ajouter des fonctionnalités lorsque votre projet sera terminé et validé.

A vous de jouer !

Accèder au projet

Pré-requis du projet :

- Pouvoir lancer du HTML, CSS et JS
- Avoir un IDE correcte (Atom, VS Code, etc...)
- Avoir la librairie p5js d'installée

Si vous ne souhaitez pas vous embeter, un IDE est disponible sur le site de p5js avec tout de déjà prêt :

> p5.js Web Editor
>
> A web editor for p5.js, a JavaScript library with the goal of making coding accessible to artists, designers, educators, and beginners.
>
> http://editor.p5js.org

Comment marche P5js ?

La librairie qui va être utilisée pour gérer tout ce qui est input et affichage se nomme P5js. En cas de soucis, la documentation officielle se trouve ici :

> reference | p5.js
>
> p5.js a JS client-side library for creating graphic and interactive experiences, based on the core principles of Processing.
>
> https://p5js.org/reference/

Il est préférable d'aller y faire un tour pour comprendre ce qui va se passer par la suite.

Un projet p5js est constitué au minimum d'un fichier HTML, d'un CSS et d'un JS nommé `sketch.js`.

Dans ce dernier se trouvent deux fonctions liées au fonctionnement de la librairie :

- `setup()` se lance une fois en début de programme et sert à initialiser les variables et lancer certaines fonctions.
- `draw()` se lance à chaque frame, c'est la boucle principale de notre jeu.

Un début de serpent

- Objectif : créer un serpent d'une case de longueur.

Commencez par créer un nouveau fichier .js (de mon côté il se nommera `snake.js`). Il va nous servir à mettre tout ce qui est en rapport avec les actions de notre serpent, que ça soit ses variables ou ses méthodes. En bref, on va initialiser l'`objet` Snake.

```
class Snake {
  constructor() {
  }
}
```

Dans la partie `constructor` se trouvera toutes les variables internes à notre serpent, le reste de la `class` servant à écrire les méthodes (ses fonctions).

> 💡 N'oubliez pas de dire à votre page html de se lancer avec ce nouveau fichier JS.

▼ Création des variables

Notre serpent étant constitué de pleins de parties, il lui faudra comme corps un tableau. Il faudra également défnir la position de la première case de ce nouveau corps à l'aide d'un vecteur. Enfin, il faudra deux variables pour donner la direction du serpent

Pas de panique, vous pouvez toujours lire la doc (CreateVector, Objet).

▼ Solution

```
constructor() {
  this.body = []
  this.body[0] = createVector(0, 0)
  this.xdir = 0
  this.ydir = 0
}
```

▼ Création de la méthode update()

Cette fonction va mettre à jour les coordonnées de la tête du serpent en lui ajoutant sa direction.

Pas de notion particulière ici, c'est à vous de vous débrouiller !

▼ Solution

```
update() {
  this.body[0].x += this.xdir
  this.body[0].y += this.ydir
}
```

▼ Création de la méthode show()

Le but ici est d'afficher la tête de notre serpent (un simple carré de dimension 1x1). Des infos se trouvent dans la doc pour vous aider (Fill, Rect, NoStroke).

▼ Solution

```
show() {
  noStroke()
  fill(0) // noir
  rect(this.body[0].x, this.body[0].y, 1, 1)
}
```

▼ Utilisation des fonctions dans sketch.js

Pour vérifier que ce que vous avez fait est foncitonnel, rendez-vous dans le fichier sketch.js.

Initilisez votre serpent dans la fonction `setup()` puis appelez les fonctions `update()` et `show()` dans `draw()`.

▼ Solution

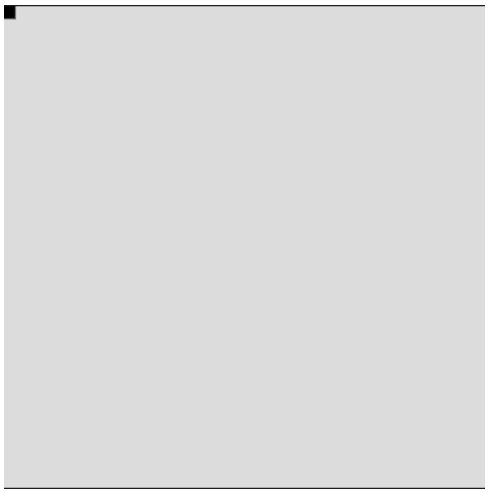
```
let snake

function setup() {
  createCanvas(400, 400)
  snake = new Snake()
}

function draw() {
  background(220)
  snake.update()
  snake.show()
}
```

Vous devriez alors appercevoir un petit point noir (ou autre couleur de votre choix) en haut à gauche de la zone de jeu (oui c'est petit, on va corriger ça). En modifier les valeurs xdir et ydir de votre serpent, vous devriez également être en mesure de faire bouger ce petit pixel.

Magnifique, vous avez maintenant un début de serpent !



Un serpent à vos ordres

- Objectif : controller le carré créé plus tôt.

Dans le fichier `snake.js` :

▼ Création de la méthode setDir(x, y)

Vous avez remarqué tout à l'heure que modifier les variables `xdir` et `ydir` faisait bouger notre serpent ? C'est le moment de s'en servir !

Vous aurez peut être remarqué la présence d'un `x` et d'un `y` dans les parametres de la fonction, il serait bon de ne pas les oublier.

▼ Solution

```
setDir(x, y) {
  this.xdir = x
  this.ydir = y
}
```

Dans le fichier `sketch.js` :

▼ Création de la fonction keyPressed()

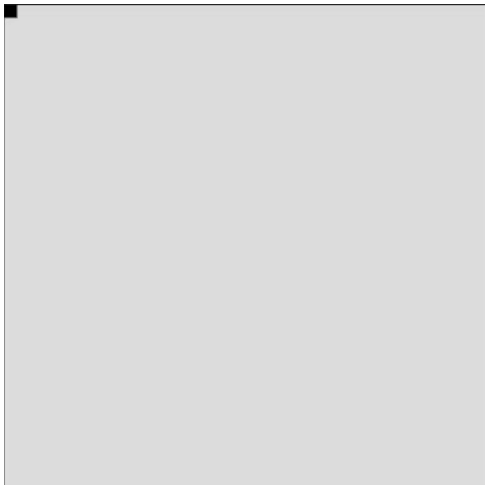
P5js possède une fonction nommée `keyPressed()` qui ,une fois appelée, permet de récupérer dans les variables `key` et `keyCode` les touches appuyées.

Pour plus d'infos, vous pouvez vous rendre ici.

▼ Solution

```
function keyPressed() {
  if (keyCode == LEFT_ARROW)
    snake.setDir(-1, 0)
  else if (keyCode == RIGHT_ARROW)
    snake.setDir(1, 0)
  else if (keyCode == DOWN_ARROW)
    snake.setDir(0, 1)
  else if (keyCode == UP_ARROW)
    snake.setDir(0, -1)
}
```

Et voilà, votre ~~pixel~~ serpent se déplace à l'écran !



Zoomer notre jeu

- Objectif : Zoomer notre jeu et modifier le framerate.

Vous avez peut être remarqué, mais on voit pas grand chose avec ce tout petit pixel de serpent. Vous pourriez contrer ça en faisant un serpent plus gros qui se déplacerait de plus d'une case à la fois, mais ici on a pas envie de s'embeter, on va directement zoomer !

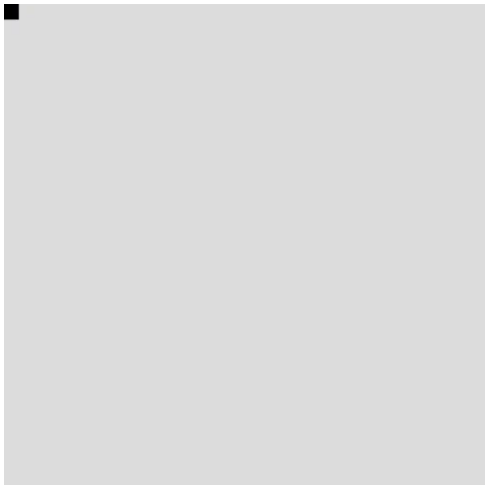
Un peu de lecture de doc pour la route ? C'est par ici.

▼ Solution

```
let rez = 20

...

funciton draw() {
  scale(rez)
  ...
}
```

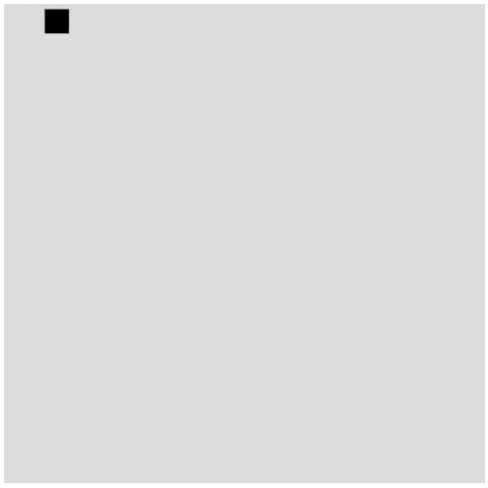


Mais c'est que ça va un peu vite par ici pour un Snake non ?

Allez hop, encore un peu de doc !

▼ Solution

```
function setup() {
  frameRate(5) // Vous pouvez mettre plus si vous le souhaitez
  ...
}
```



Un petit casse-croute

- Objectif : Générer de la nourriture sur la map

Maintenant que notre serpent se déplace, il va falloir créer sa nourriture et la placer sur le terrain.

Pour rester dans l'orienté objet, la nourriture en sera un elle aussi.

Pour commencer, créez un nouveau fichier (`food.js` par exemple) et initialisez un nouvel objet Food.

▼ Solution

```
class Food {
  constructor() {
  }
}
```

N'oubliez pas de dire à votre page html de se lancer avec ce nouveau fichier JS.

▼ Création de ses variables

La nourriture aura comme variable :

- Une coordonnée `X` aléatoire.
- Une coordonnée `Y` aléatoire.
- Un `corps` placé au niveau de son `X` et `Y`.

N'hésitez pas à retourner voir comment était fait l'objet Snake. Un peu de lecture de doc ne fera pas de mal non plus (floor, random, dimensions).

▼ Solution

```
constructor() {
    this.x = floor(random(width / rez))
    this.y = floor(random(height / res))
    this.body = createVector(this.x, this.y)
  }
```

▼ Création de la méthode show()

Sur le même principe que la fonction show() du serpent, vous devez coder celle du fruit.

▼ Solution

```
show() {
  noStroke() // Supprime les bordures noires
  fill(255, 0, 0) // rouge
  rect(this.body.x, this.body.y, 1, 1)
}
```

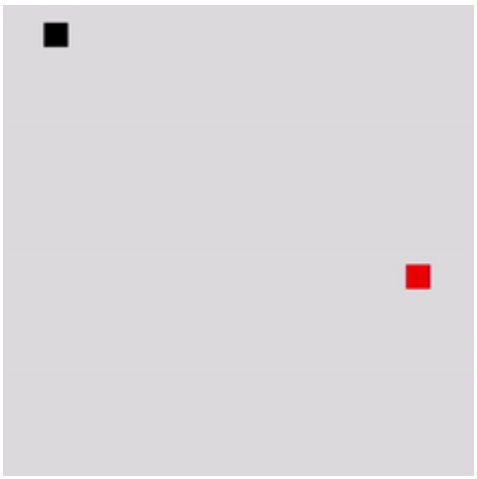
Pour terminer, vous pouvez initialiser un fruit dans le fichier `sketch.js` et l'afficher à l'aide de votre nouvel objet.

▼ Solution

```
let food

function setup() {
  ...
  food = new Food()
}

function draw() {
  ...
  food.show()
}
```



Miam Miam

- Objectif : Manger les pixels rouges.

On a un serpent, on a de la nourriture, maitenant il faut que le serpent mange cette nourriture !

Dans le fichier `snake.js` :

▼ Création de la méthode eat(food)

La méthode `eat()` va vérifier si la tête du serpent (donc la première case de `body`) se trouve aux même cordonnées `x` et `y` que la nourriture `food`.

Si oui, il faut retourner `true`, si non `false`.

▼ Solution

```
eat(food) {
  let head = this.body[0]

  if (head.x == food.x && head.t == food.y)
    return true
  return false
}
```

Dans le fichier `sketch.js` :

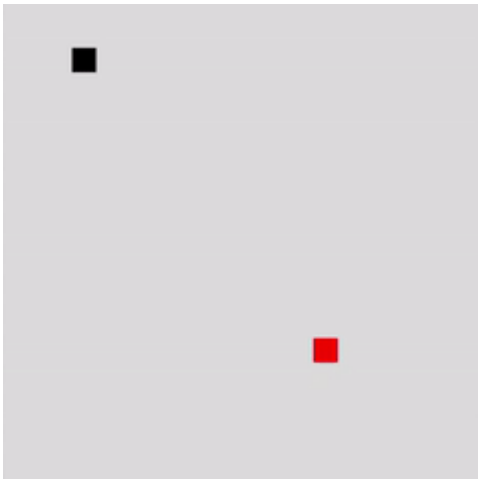
▼ Génération d'une autre nourriture si le serpent a mangé la dernière

Si `eat()` retourne la valeur `true`, alors il faut créer une nouvelle instance de Food.

▼ Solution

```
function draw() {
  ...
  if (snake.eat(food))
    food = new Food()
  ...
}
```

Bravo, votre serpent peut désormait manger de la nourriture à volonté !



C'est qu'il a mangé de la soupe !

- Objectif : Faire grandir le serpent à chaque fois qu'il mange.

Toute cette partie se déroule dans le fichier `snake.js` et va modifier certaines fonctions créées plus tôt dans le sujet (déso).

▼ Création d'une nouvelle variable `len`

Pour faciliter l'écriture des prochaines lignes de code, je conseil de créer une variable len, interne à Snake, et servant à connaitre la longueur du serpent.

▼ Solution

```
constructor() {
  ...
  this.len = 1
}
```

▼ Création de la méthode grow()

La méthode grow() permet d'augmenter len, puis de copier le dernier element du corps du serpent pour l'ajouter à body.

Un peu de doc pour la route : Push.

Pour copier un element, vous pouvez utiliser la méthode `copy()`.

▼ Solution

```
grow() {
  let head = this.body[this.len - 1].copy();
  this.len++;
  this.body.push(head);
}
```

Pour finir, n'oubliez pas d'executer cette méthode quand un fruit est mangé.

▼ Solution

```
eat(food) {
  ...
  if (...) {
    this.grow()
    return true
  }
  return false
}
```

▼ Modification de show()

Au lieu de n'afficher que l'élément 0 de body, il faut maintenant faire en sorte d'afficher tous les éléments du tableau.

Un peu de documentation.

▼ Solution

```
show() {
  for (let i = 0; i != this.len; i++) {
    noStroke()
    fill(0)
    rect(this.body[i].x, this.body[i].y, 1, 1)
  }
}
```

▼ Modification de update()

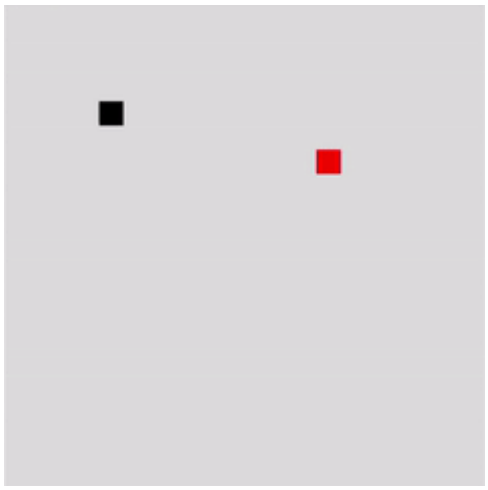
Ici, on aura besoin des méthodes shift, copy et push.

Il va falloir récupérer le dernier élément du corps du serpent, shifter la liste, modifier les coordonnées et push.

▼ Solution

```
update() {
  let head = this.body[this.len - 1].copy();
```

```
  this.body.shift()
  head.x += this.xdir;
  head.y += this.ydir;
  this.body.push(head);
}
```



Et hop, un serpent qui grandit. Par contre il va falloir aller modifier la fonction eat, car on mange avec les fesses (je suis pas certain que ça marche comme ça).

▼ Modification de la fonction eat()

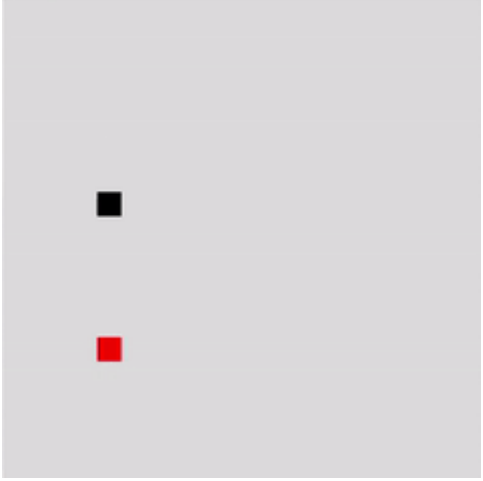
Comme vous avez pu le tester plus tôt, le serpent mange avec les fesses. Et c'est normal (enfin non mais vous m'avez compris) car la partie du corps servant à `eat()` est celle en case 0, soit le bout de la queue du serpent.

A vous de trouver comment lui donner la tête, qu se trouve à l'autre bout de la liste !

▼ Solution

```
eat(food) {
  let head = this.body[this.len - 1]
  ...
}
```

Bravo, votre serpent peut maintenant grandir ! Plus qu'une étape avant de terminer votre Snake !



Puisse le sort vous être favorable

- Objectif : Game Over du joueur.

Dans le fichier `snake.js` :

▼ Détecter la défaite avec la méthode endGame()

La défaite se déclenche si la tête du serpent sort de l'écran ou si elle touche une autre partie de son corps.

Vous pouvez donc vérifier les coordonnées de la tête avec dans un premier temps la hauteur et la largeur, puis, avec une boucle, si les coordonnées sont les même que celle d'une partie du corps.

N'oubliez pas de retourner `true` ou `false`.

▼ Solution

```
endGame() {
  let x = this.body[this.len - 1].x
  let y = this.body[this.len - 1].y
  let w = floor(width / rez)
   let h = floor(height / rez)

  if (x > w - 1 || x < 0 || y > h - 1 || y < 0)
    return true
  for (let i = 0; i < this.len - 1; i++) {
    let part = this.body[i]
    if (part.x == x && part.y == y)
      return true
  }
  return false
}
```

Dans le fichier sketch.js

▼ Arrêter le jeu en cas de défaite

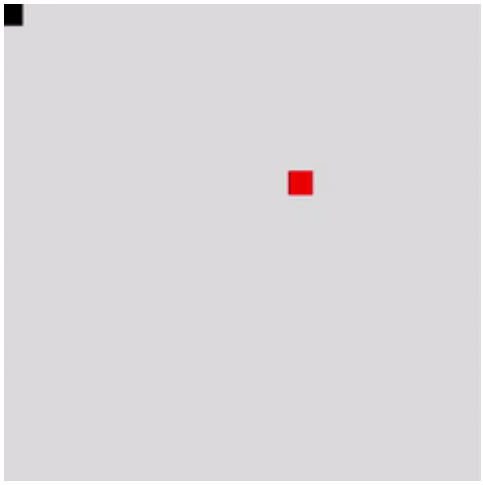
Ici, il vous faudra lancer certaines options pour stopper le jeu si la fonction endGame() vous retourne true.

Vous pouvez faire ce que vous voulez, j'ai personnelement utiliser les fonction noLoop, background et print.

▼ Solution

```
function draw() {
  ...
  if (snake.endGame()) {
    print("ENDGAME");
    background(255, 0, 0);
    noLoop();
  }
}
```

Bravo à vous, votre Snake est maintenant 100% opérationnel ! A vous les heures d'amusement !



Conclusion

"FéliSSSsssSSSitaSSSsssSSSSion, graSSSsssSSSe à vous j'ai pu manger autant de cookies que je le voulait !", vous annonce Snaky, heureux. Maintenant, il va

pouvoir passer une bonne journée dans sa maison.

Pour aller plus loin

Si vous avez terminé plus tôt que prévu, voici quelques idées pour aller plus loin :

- Ajouter un système de score qui s'affiche à l'écran ou dans la console.
- Changer les couleurs du serpent.
- Ajouter des graphismes plus jolis.
- Avoir un vrai écran de fin.