

**Score : 13**

# Candy Crush

Javascript



# Candy Crush

- Objet graphique (Sprite)
- Classe LutinBonbon
  - Sert à afficher un bonbon
  - Constructor(*quel*, largeur, hauteur)
    - *quel* est le numéro du bonbon (1, 2, 3, 4, 5)
  - positionXY(x1,y1)
    - Change la position du lutin
  - marque(v)
    - Vrai/faux : si vrai alors quand on dessine le bonbon on dessine un cadre autour

# Candy Crush

- Objet graphique (Sprite)
- dessin(contexte)
  - Dessine le sprite à sa position courante
  - Et dessine éventuellement un cadre s'il est marqué
- On souhaite gérer le déplacement en animation
- De la position courante vers une position 2
- On rajoute une position cible pour l'animation
- déplacementVers(x2,y2)

# Candy Crush

- Si la position cible n'est pas atteinte, alors le lutin est considéré comme
  - `estEnMouvement()` // vrai
- Il faut le faire un peu avancer
  - Donc modifier ses coordonnées

```
metAJour() { // met à jour les coordonnées du sprite si besoin
  if (! this.estEnMouvement() ) return // pas la peine
  if ( (this.x != this.versX) || (this.y != this.versY) ) {
    if (this.versX > this.x) this.x = this.x +5
    if (this.versX < this.x) this.x = this.x -5
    if (this.versY > this.y) this.y = this.y +5
    if (this.versY < this.y) this.y = this.y -5
  } else {
    if (this.estEnMouvement()) {
      this.bouge = false;
    }
  }
}
```

# Candy Crush

- Donc pour animer un lutin
  - `var bonbon = new LutinBonbon(1, 40, 40)`
  - `bonbon.position(20,20)` // force la position mais n'affiche rien
  - `bonbon.deplacementVers(x2,y2)` // ne déplace rien, indique la cible à atteindre
  - `bonbon.metAJour()` // déplace un peu la position courante
  - `bonbon.dessin(cahier)` // dessine le sprite en position (effacer écran ?)
  - `bonbon.metAJour()` // déplace un peu la position courante
  - `bonbon.dessin(cahier)` // dessine le sprite en position
  - `bonbon.metAJour()` // déplace un peu la position courante
  - `bonbon.dessin(cahier) ....`
  - .... On arrête quand le sprite a sa position finale

# Candy Crush

- Donc pour animer un lutin
  - `var bonbon = new LutinBonbon(1, 40, 40)`
  - `bonbon.position(20,20) // force la position mais n'affiche rien`
  - `bonbon.deplacementVers(x2,y2) // ne déplace rien, indique la cible à atteindre`
  - `do {`
    - `bonbon.metAJour() // déplace un peu la position courante`
    - `bonbon.dessin(cahier) // dessine le sprite en position (effacer écran )`
  - `} while (bonbon.estEnMouvement())`

IMPOSSIBLE

# Candy Crush

- En Javascript
  - Asynchrone

```
var bonbon = new LutinBonbon(1, 40, 40)
bonbon.position(20,20)
bonbon.deplacementVers(x2,y2)
anime() // fin des instruction du programme
// pas d'instruction en dessous
```

```
function anime(cahier) {
    bonbon.metAJour()
    bonbon.dessin(cahier)
    if (bonbon.estEnMouvement())
        setTimeout(anime, 100)
}
```

ASYNCR

# Candy Crush

- La Vue va contenir tous les sprites
  - Sans doute dans un tableau de sprites
- Chaque sprite/lutin de la vue a ses propres coordonnées (c'est un objet)
- Chaque sprite de la vue peut être en mouvement ou pas
- Quand on affiche la vue, on demande à chaque sprite de s'afficher
- Quand on demande à la vue de se mettre à jour, on demande à chacun de ses sprites de se remettre à jour
  - Un par un
  - PUIS on redessine toute la vue !



# Candy Crush : pour la VUE

```
dessineTout() { // Vue
    effaceEcran()
    pour chaque sprite
        sprite.dessin(cahier)
    fpour
}

metAJour() { // Vue
    pour chaque sprite
        sprite.mettreAJour()
    fpour
}

estEnMouvement() { // Vue
    Vrai si au moins un sprite de la vue est en mouvement
}
```

# Candy Crush

```
var vue = new Vue( ..... )  
// on rajoute des sprites à la vue  
// ....  
// on demande des déplacements à faire à certains sprites  
animeVue()  
  
function animeVue(cahier) {  
    vue.metAJour() // itere sur la mise à vue de chaque Sprite  
    vue.dessinTou(cahier)  
    if (vue.estEnMouvement())  
        setTimeout(animeVue, 10)  
}
```

# Candy Crush

```
var vue = new Vue( ..... )  
// on rajoute des sprites à la vue  
// ....  
// on demande des déplacements à faire à certains sprites  
vue.animeVue()
```

```
function animeVue(cahier) { // Méthode de Vue  
  this.metAJour() // itere sur la mise à vue de chaque Sprite  
  this.dessinTou(cahier)  
  if (this.estEnMouvement())  
    setTimeout(this.animeVue, 10)  
}
```

IMPOSSIBLE

# Candy Crush

```
var vue = new Vue( ..... )  
// on rajoute des sprites à la vue  
// ....  
// on demande des déplacements à faire à certains sprites  
vue.animeVue()  
// class Vue..... {  
// méthode de Vue  
function animeVue(contexte) {  
    this.metAJour() // itere sur la mise à vue de chaque Sprite  
    this.dessin(contexte)  
    var that = this  
    if (this.estEnMouvement())  
        setTimeout( () => { that.animeVue(contexte) }, 10) }  
}  
} //
```

# Candy Crush et MVC

- Modèle :
  - État de la grille de bonbon
  - Indépendant de l'affichage
    - Donc indépendant des animations (qui ne concernent que la vue)
  - 5 types de bonbons
  - Le Score (stocké, pas d'affichage)
- Modèle : pensez à la sauvegarde / save / reload
  - Qu'est-ce qu'on met dedans ?
  - Ce qu'on y met, c'est le modèle
  - Donc pas l'animation etc..

# Candy Crush et MVC

- Vue :
  - A minima, la vue doit pouvoir afficher le modèle
  - Mais elle peut faire plus que ça !
  - Animation secondaires, décorations, thème
  - Sons, bruitages
- La vue contient clairement les sprites bonbon
- Dans le cas de ce problème, un tableau de sprites
- Pourquoi ne pas utiliser le modèle et éviter de refaire un tableau de sprites ?
  - Parce que l'animation a besoin d'informations non cruciales en plus
  - Que chaque bonbon peut être animé indépendamment
  - Parce que l'animation et le modèle peuvent ne pas être synchronisés

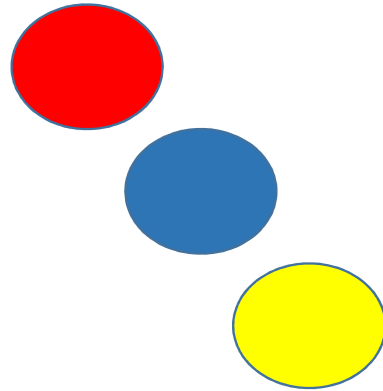
# Candy Crush et MVC

- Contrôleur :
  - Le chef d'orchestre des évènements
  - Il se déclenche sur des événements
  - Click souris : évènement évident
  - Le click souris va déclencher la réaction du modèle et de la vue
  - Il faut interdire les clicks souris pendant les phases
    - De stabilisation de la grille
    - Et donc d'animation
  - Donc les évènements à prendre en compte
    - Click + fin d'animation

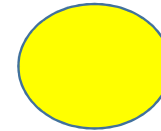
# Candy Crush et MVC

- En théorie

- 3 objets
  - Un modèle
  - Une vue
  - Un contrôleur

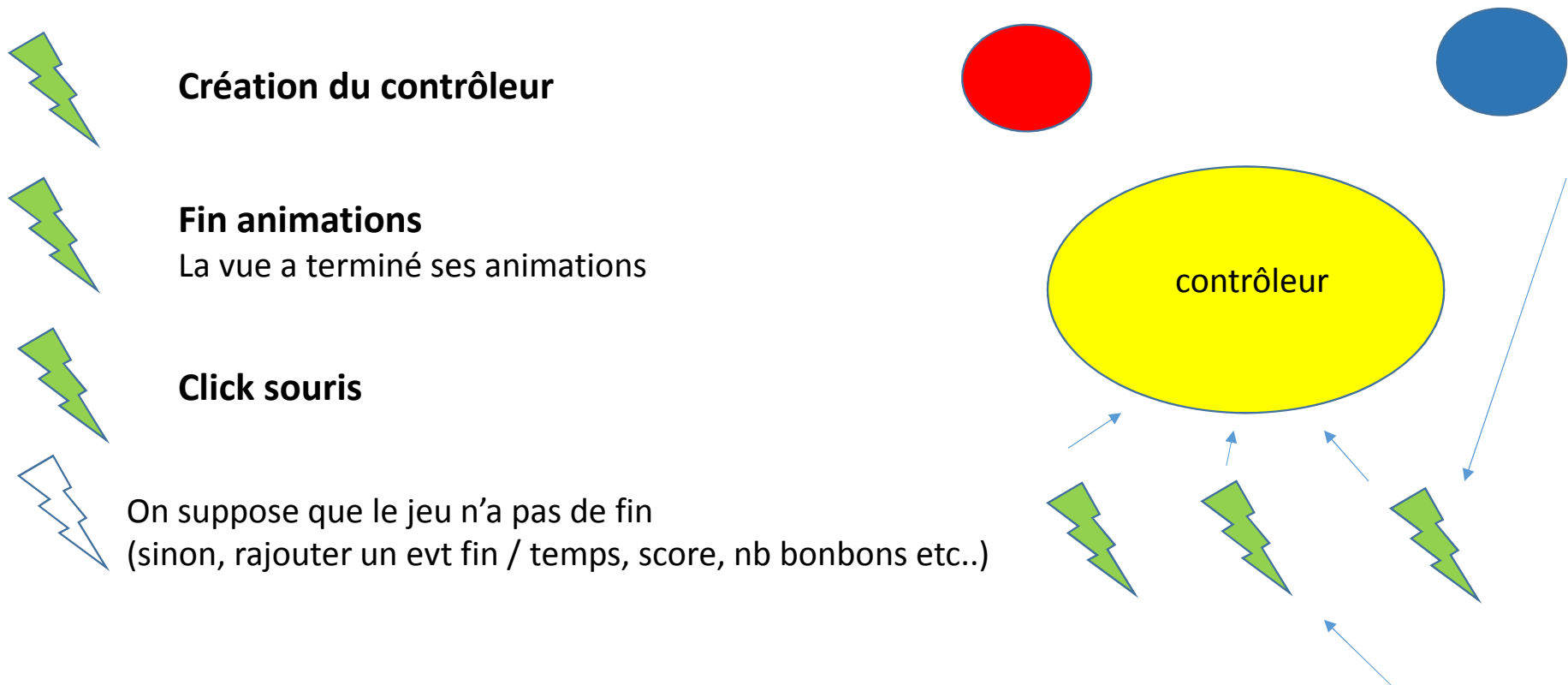


- Ici le contrôleur est le chef d'orchestre
  - Le contrôleur crée un modèle et une vue





Le contrôleur répond à des évènements  
En réponse, il demande au **modèle** de se modifier  
Et à la **vue** de se mettre à jour



Modèle

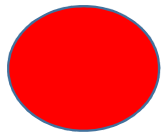


Tableau d'entiers ou « » si vide  
Entier pour le score  
Création : grille au hasard (sans trous)

Vue



Tableau de lutins (vide au départ)  
Mission : affiche les lutins et le score

new

Accès lecture

new

La vue demande au modèle son contenu  
Crée des lutins qui correspondent aux  
Codes bonbon (1 -> rouge, 2 -> vert..)

**Création du contrôleur (méthode)**

`vue.metAJourAPartirModele()`

Création du modèle

**puis** Création de la vue

**puis** Demande à la vue de se mettre à jour / modèle

**puis** Lance l'animation de la vue

Cf animation

`vue.animeVue()`

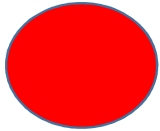
`setTimeout...`

**STOP**

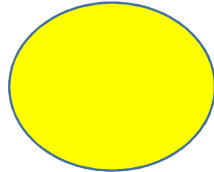


Evt fin animation

Modèle



Contrôleur



Vue



Attention gestion score :  
bloquer jq pas trous + filtrage click



## Fin animation

la vue est synchronisée sur le modèle (rien ne bouge plus)  
modele.explosionPossible() ? on demande au modèle s'il y a des alignements ?



→ OUI ! On demande au modèle de les faire exploser

**puis** on demande à la vue de se mettre à jour **puis** animation

**STOP**



vue.metAJourAPartirModele()

vue.animeVue()



→ NON ! Le contrôleur appelle une méthode **repackGrille** qui  
regarde et corrige dans le modèle les trous en comblant les colonnes  
verticalement : à chaque trou comblé, il demande à la vue de créer un lutin  
correspondant dont la cible doit être la position finale.

puis animation de la vue (pas de mise à jour ici, c'est le contrôleur qui)

**STOP**



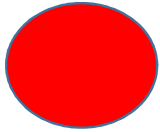
vue.animeVue()



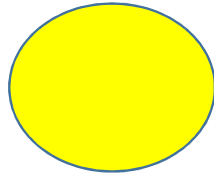
Pas de trous  
Jeux + score

accès

Modèle



Contrôleur



Vue



**Click souris** (var) pas d'evt tant que lancement pas terminé

(var) On détecte le premier / second click

**Premier click** : on note la case sélectionnée **puis** on indique à la vue que le lutin de cette case est marqué **puis** on demande l'animation de la vue

**STOP**



vue.marqueLutin(x,y)

vue.animeVue()



**Second click** : on prend seulement si différent case courante, voisin

échangeCases(x1,y1,x2,y2) ? On demande au modèle d'échanger les 2 cases (on ne touche pas à la vue)

**Puis** On demande au modèle si explosion possible ?

vue.enleveMarqueLutin(x,y)

**NON** : on demande du modèle de rééchanger les deux cases (on remet dans l'ordre)

**Puis** désélection dans la vue **puis** affichage (ou animation)

vue.animeVue()

vue.afficheVue()



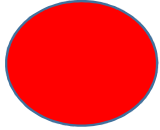
**STOP**



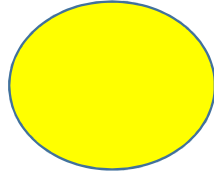
échangeCases(x1,y1,x2,y2) ?

explosionPossible() ?

Modèle



Contrôleur



Vue



**Click souris** (var) pas d'evt tant que lancement pas terminé

(suite)

OUI : indiquer l'échange des deux lutins à la vue **puis** animation de la vue

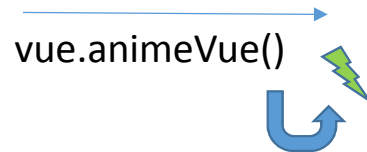
**STOP**



Vue.echangeLutins(x1,y1,x2,y2)



vue.animeVue()



# Organisation objet

```
class LutinBonbon { //

    constructor(quel, largeur, hauteur ) {
        // quel : type de l'objet (1, 2, 3, 4, 5) - en gros un tag sur le bonbon
        // largeur, hauteur : zoom à l'affichage
    }

    positionXY(x1, y1) { // change la position
    }

    typeBonbon() { // renvoie le type du sprite
    }

    marque(v) { // true ou false : indique si l'objet est selectionné ou pas
    }

    dessin(context) { // dessine le sprite avec sa position courante
    }
}
```

# Organisation objet

```
deplacementVers(x2, y2) { // prevoit un déplacement vers x2,y2  
}
```

```
estEnMouvement() { // renvoie vrai si le sprite bouge encore  
}
```

```
metAJour() { // met à jour les coordonnées du sprite si besoin  
}
```

```
} // class LutinBonbon
```

# Organisation objet

```
class Vue { // contient le modèle de la vue

    constructor(txy, monControleur, modele, tailleLutin) { // on passe la taille et un objet contrôleur en argument
    }

    metAJourAPartirDuModele() { // la vue va aller chercher son état dans son propre modèle
    }

    nouveauLutin(x,y, quel) {
    }

    echange2lutins(x1,y1,x2,y2) { // intervertit deux cases dans la vue
    }

    afficheVue(contexte) { // dessine la vue sans animation
    }

    animeVue(contexte) { // anime les bonbons et quand c'est terminé, appelle le contrôleur
    }
}
```



# Organisation objet

```
class Modele { //

    constructor(taille) { // taille est le coté du carré en nombre de cases
    }

    echange2cases(x1,y1,x2,y2) { // deux cases dans le modèle
    }

    faitExplosion() { // on fait les explosions du modèle
    }

    explosePossible() { // est-ce qu'il y a des explosions potentielles dans le modèle ?
    } // explose

}
```

# Organisation objet

```
class Controleur { //
```

```
    constructor(tailleJeu, tailleLutin) {  
    }
```

```
    finAnimation(contexte) { // est appelé automatiquement quand la vue s'est mise à jour avec l'animation  
        // Quand la vue s'est stabilisée, ca veut dire que l'animation en cours  
        // est terminée et que la vue reflète bien le modèle  
        // à ce stade, il faut pour le contrôleur, observer le modèle et le modifier  
        // et éventuellement relancer une animation de la vue
```

```
    }
```

```
    click(x,y) { // on vient de cliquer à la position x,y dans l'écran
```

```
    } // click(x,y)
```

# Organisation objet

```
repackGrille(contexte) { // fait tomber et rebouche les trous en créant de nouveaux bonbons  
}
```

```
repackColonne(col) { // repack une colonne: donne faux si pas besoin  
} // repackColonne
```

```
}
```

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <script src="mscript.js"></script>
    <title>test</title>
  </head>
  <body onload="init()">
    <canvas id="cvs" width="500" height="500" style="margin: 10px auto; border: solid 1px #000;"></canvas>
    <img id="button" onclick="init()">
  </body>
</html>
```

```
function captureClick(event) // on intercepte le click souris
{ // calcul des coordonnées de la souris dans le canvas
  if (event.target.id == "cvs") {
    var x = event.pageX - event.target.offsetLeft;
    var y = event.pageY - event.target.offsetTop;
    jeu.click(x,y)
  }
}
```

```
function init() { // démarrage du jeu
  // variable globale
  context = document.getElementById("cvs").getContext("2d");
  context.width = document.getElementById("cvs").width;
  context.height = document.getElementById("cvs").height;

  document.addEventListener("click", captureClick);

  var jeu = new Controleur(10, 50) // cree une grille de 10x10 avec des lutins de taille 50x50
    // le controleur cree le modele et la vue

  jeu.maVue.metAJourAPartirDuModele(); jeu.maVue.animeVue(context)
}
```