OOP DESIGN WITH GAME

역할 책임 인식

역할을 객체로 책임을 메소드로

역할이란 책임의 집합

역할을 객체로 책임을 메소드로

역할이란 책임의 집합

책임은 다른 역할과의 관계로 발생

메소드는 수신자, 송신자, 메세지로 구성됨

Item

모델

Game

컨트롤러

Item 

Game

모델

컨트롤러

Item ⟷ Game

도메인분석

모델

```
type,x,y,
selected,pre,
isActivated

pos(x,y)

isBorder(item)

select(item)
unselect()
isSelected(item)

action()
queAction():P
```

컨트롤러

```
removeItem(item)
addQue(item)
getBorder(item)
```

Item 

Game 

Factory

모델

type,x,y,
selected,pre,
isActivated

pos(x,y)

isBorder(item)

select(item)
unselect()
isSelected(item)

action()
queAction():P

컨트롤러

removeItem(item)
addQue(item)
getBorder(item)

모델팩토리

getRenderer(renderer)
getItem(game, type, c, r)
get extra()
setPriority(rc, cc)
reset(){}

Item ↔ Game ↔ Factory

도메인분석 ↔ 도메인분석

모델

```
type,x,y,
selected,pre,
isActivated

pos(x,y)

isBorder(item)

select(item)
unselect()
isSelected(item)

action()
queAction():P
```

컨트롤러

```
removeItem(item)
addQue(item)
getBorder(item)
```

Msg

Renderer

렌더러

모델팩토리

```
getRenderer(renderer)
getItem(game, type, c, r)
get extra()
setPriority(rc, cc)
reset(){}
```

SubRenderer

서브렌더러

```
const Item = class{
   //일반기능
   get type(){}
   get x(){}
   get y(){}
   //셀렉트관련
   get isSelected(){}
   get previousSelected(){}
   //액션관련
   get get isActionActivated(){}
}
```

```
const Item = class{
   …
   isBorder(item){
      const white = {item};
      if(!white.item) return err(`invalid item:${item}`);
      const {item:{x:ix, y:iy}} = white, {x:tx, y:ty} = this;
      return this != white.item  &&  Math.abs(ix-tx)<2  &&  Math.abs(iy-ty)<2;
   }
}
```

```
const Item = class{
   …
   isBorder(item){
      const white = {item};
      if(!white.item) return err(`invalid item:${item}`);
      const {item:{x:ix, y:iy}} = white, {x:tx, y:ty} = this;
      return this != white.item && Math.abs(ix - tx) < 2 && Math.abs(iy - ty) < 2;
   }
   isSelectedList(item){
      const {_previousSelected:prev} = this;
      if(!prev) return false;
      if(prev == item) return true;
      return prev.isSelectedList(item);
   }
}
```

```
let isDebug = true; //false
Const err = msg=>{
   if(!isDebug){
      console.log(msg);
      return false;
   }else{
      throw msg;
   }
}
```

```
let debugMode = 'debug'; //debug, log, none
const err = msg=>{
    switch(debugMode){
    case'log':
        console.log(msg);
        return false;
    case'none':return false;
    default:throw msg;
    }
}
```

```
const Item = class{
   …
   isBorder(item){
      const white = {item};
      if(!white.item) return err(`invalid item:${item}`);
      const {item:{x:ix, y:iy}} = white, {x:tx, y:ty} = this;
      return this != white.item && Math.abs(ix - tx) < 2 && Math.abs(iy - ty) < 2;
   }
   isSelectedList(item){
      const {_previousSelected:prev} = this;
      if(!prev) return false;
      if(prev == item) return true;
      return prev.isSelectedList(item);
   }
   setPos(x, y){
      this.x = x, this.y = y;
   }
}
```

```
const Item = class{
   …
   isBorder(item){
      const white = {item};
      if(!white.item) return err(`invalid item:${item}`);
      const {item:{x:ix, y:iy}} = white, {x:tx, y:ty} = this;
      return this != white.item && Math.abs(ix - tx) < 2 && Math.abs(iy - ty) < 2;
   }
   isSelectedList(item){
      const {_previousSelected:prev} = this;
      if(!prev) return false;
      if(prev == item) return true;
      return prev.isSelectedList(item);
   }
   setPos(x, y){
      this.x = x, this.y = y;
   }
   select(previousItem){
      this._isSelected = true;
      this._previousSelected = previousItem;
   }
}
```

```
const Item = class{
   constructor(_game, _type, _x, _y){
      if(!_game) return err(`invalid game:${game}`);
      prop(this, {
         _game, _type, _x, _y,
         _isSelected:false, _previousSelected:null,
         _isActionActivated:false
      });
   }
}
```

```
let debugMode = 'debug'; //debug, log, none
const err = msg=>{
    switch(debugMode){
    case'log':
        console.log(msg);
        return false;
    case'none':return false;
    default:throw msg;
    }
}
const Item = class{
    constructor(_game, _type, _x, _y){
        if(!_game) return err(`invalid game:${game}`);
        prop(this, {
            _game, _type, _x, _y,
            _isSelected:false, _previousSelected:null,
            _isActionActivated:false
        });
    }
    get type(){}
    get x(){return this._x;}
    get y(){}
    isBorder(item){
        const white = {item};
        if(!white.item) return err(`invalid item:${item}`);
        const {item:{x:ix, y:iy}} = white, {x:tx, y:ty} = this;
        return this != white.item  &&  Math.abs(ix-tx) < 2 &&
               Math.abs(iy - ty) < 2;
    }
    ...
    ...
    setPos(x, y){
        this.x = x, this.y = y;
    }
    get isSelected(){}
    get previousSelected(){}
    isSelectedList(item){
        const {_previousSelected:prev} = this;
        if(!prev) return false;
        if(prev == item) return true;
        return prev.isSelectedList(item);
    }
    select(previousItem){
        this._isSelected = true;
        this._previousSelected = previousItem;
    }
    unselect(){
        this._isSelected = false;
        this._previousSelected = null;
    }
    get isActionActivated(){}
    action(){return false;}
    queAction(){}
}
```