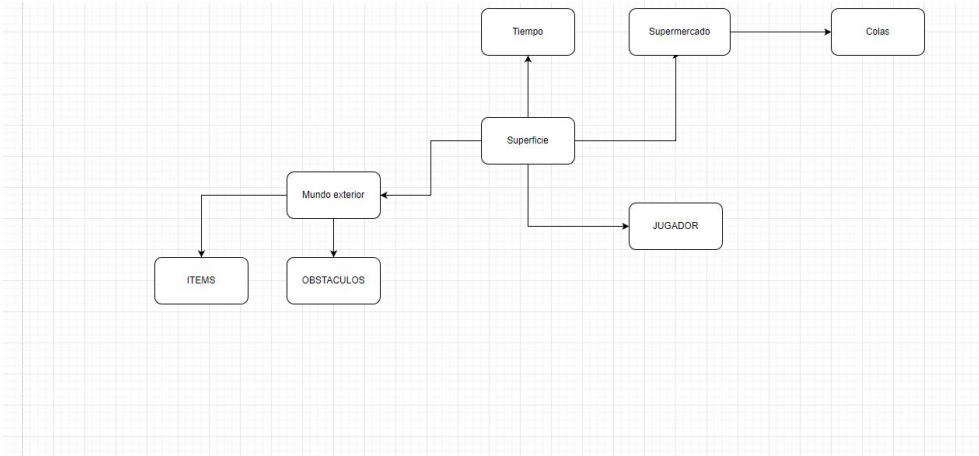
Proyecto Laberinto-Colas

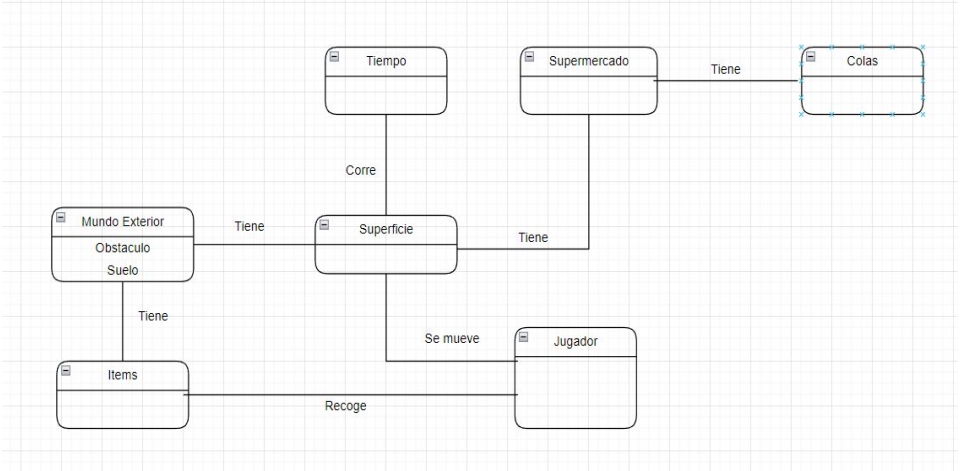
Modelo de dominio:

Primera Versión



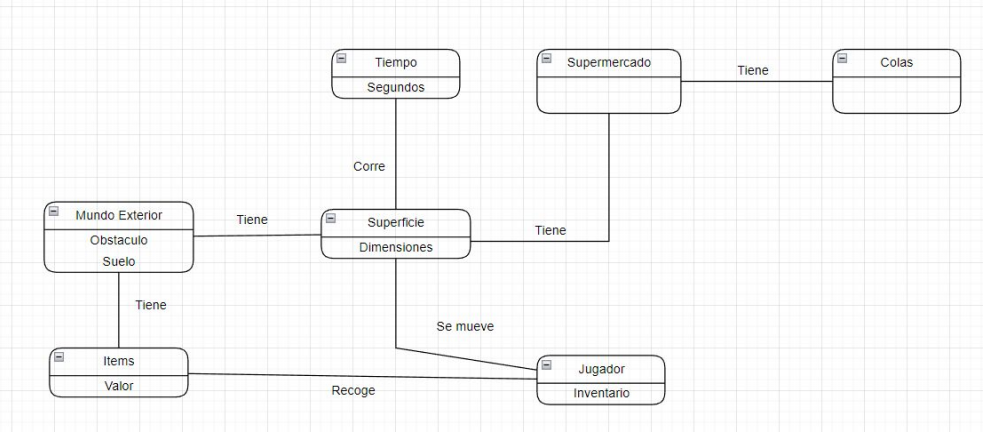
Modelo de dominio:

Segunda Versión



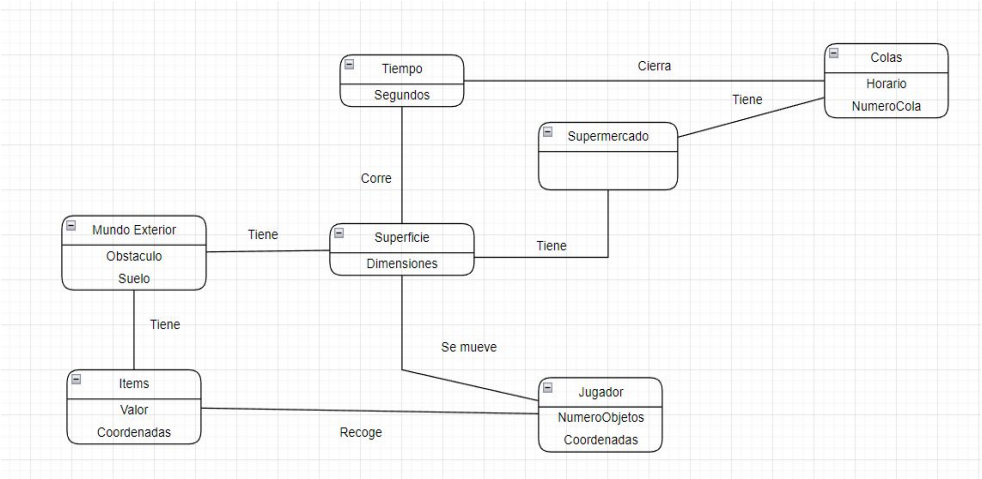
Modelo de dominio:

Tercera Versión



Modelo de dominio:

Cuarta Versión



Modelo de dominio:

Versión Final

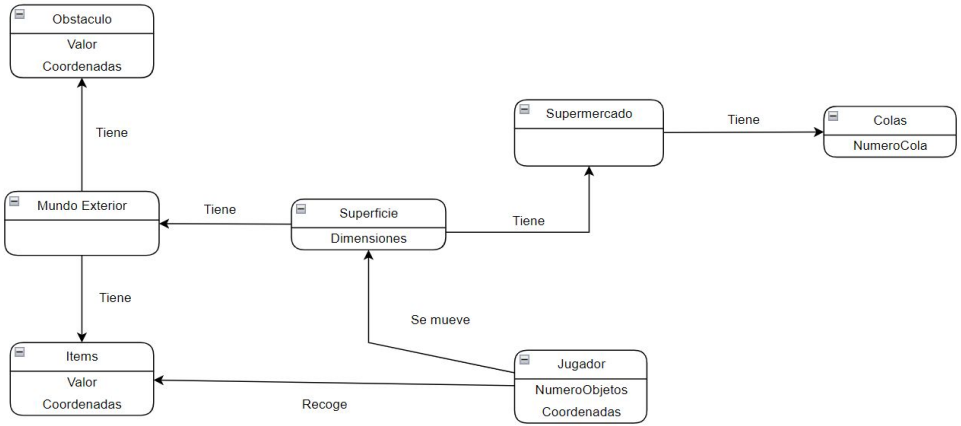


Diagrama de Clases:

Primera Versión

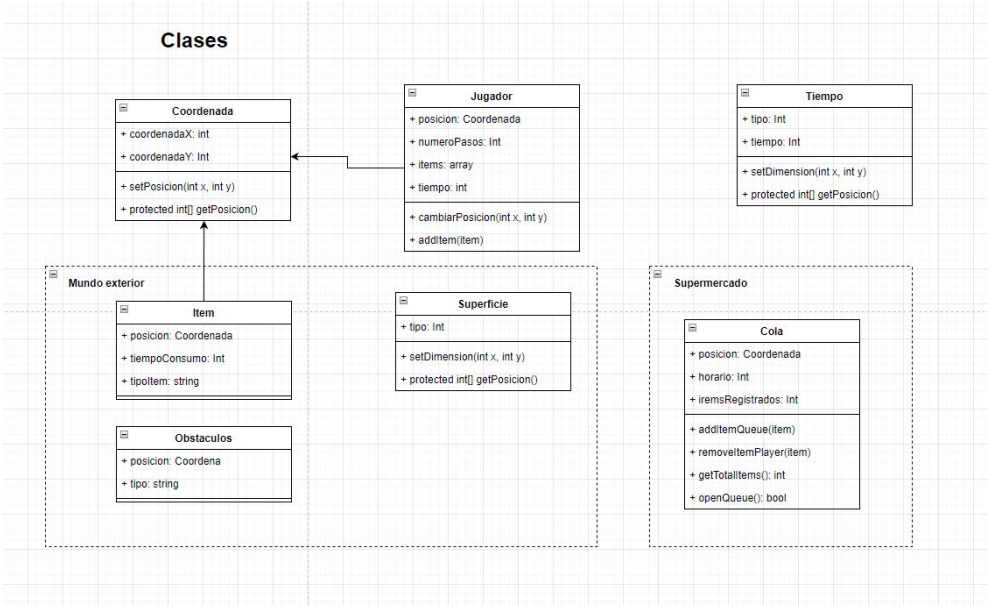


Diagrama de Clases:

Segunda Versión

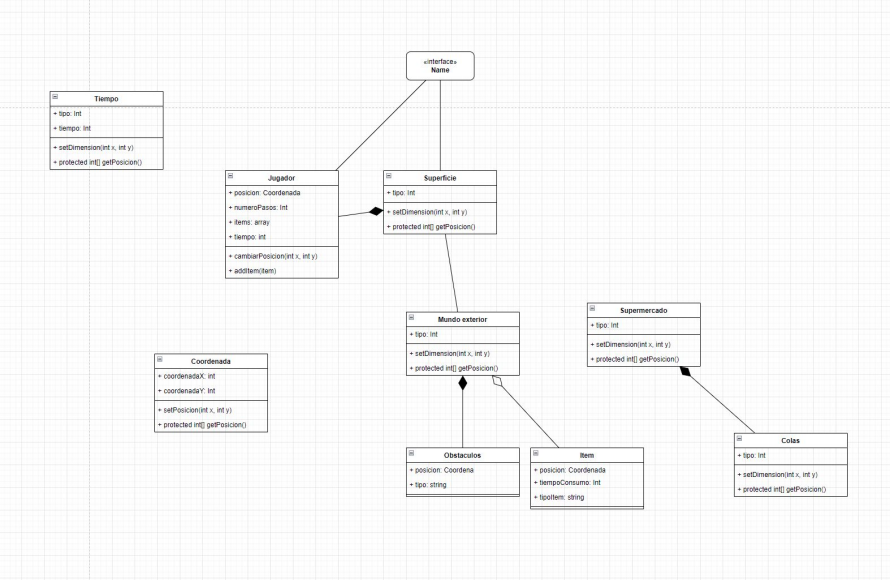
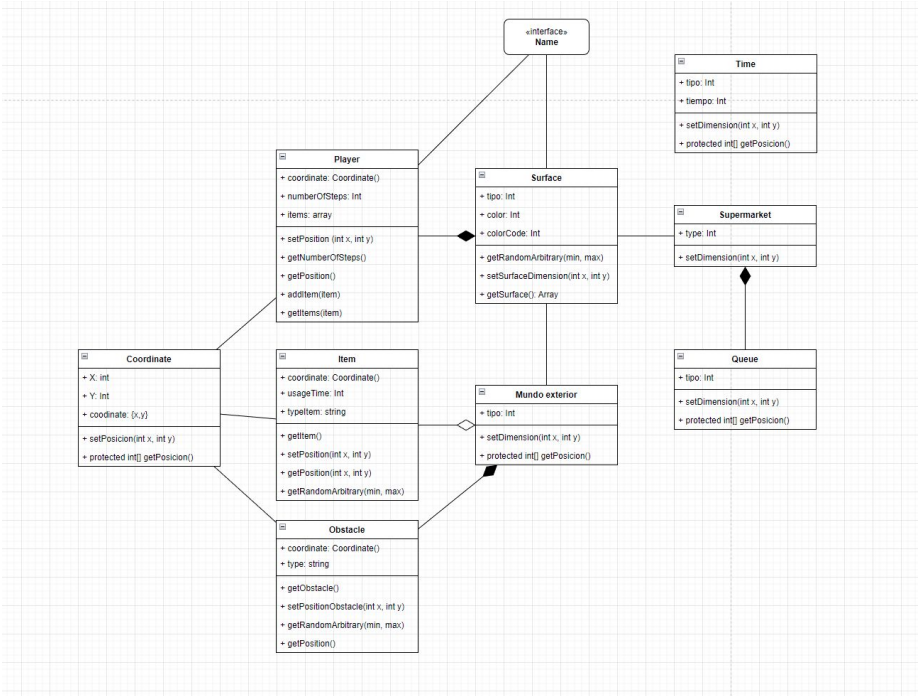


Diagrama de Clases:

Tercera Versión



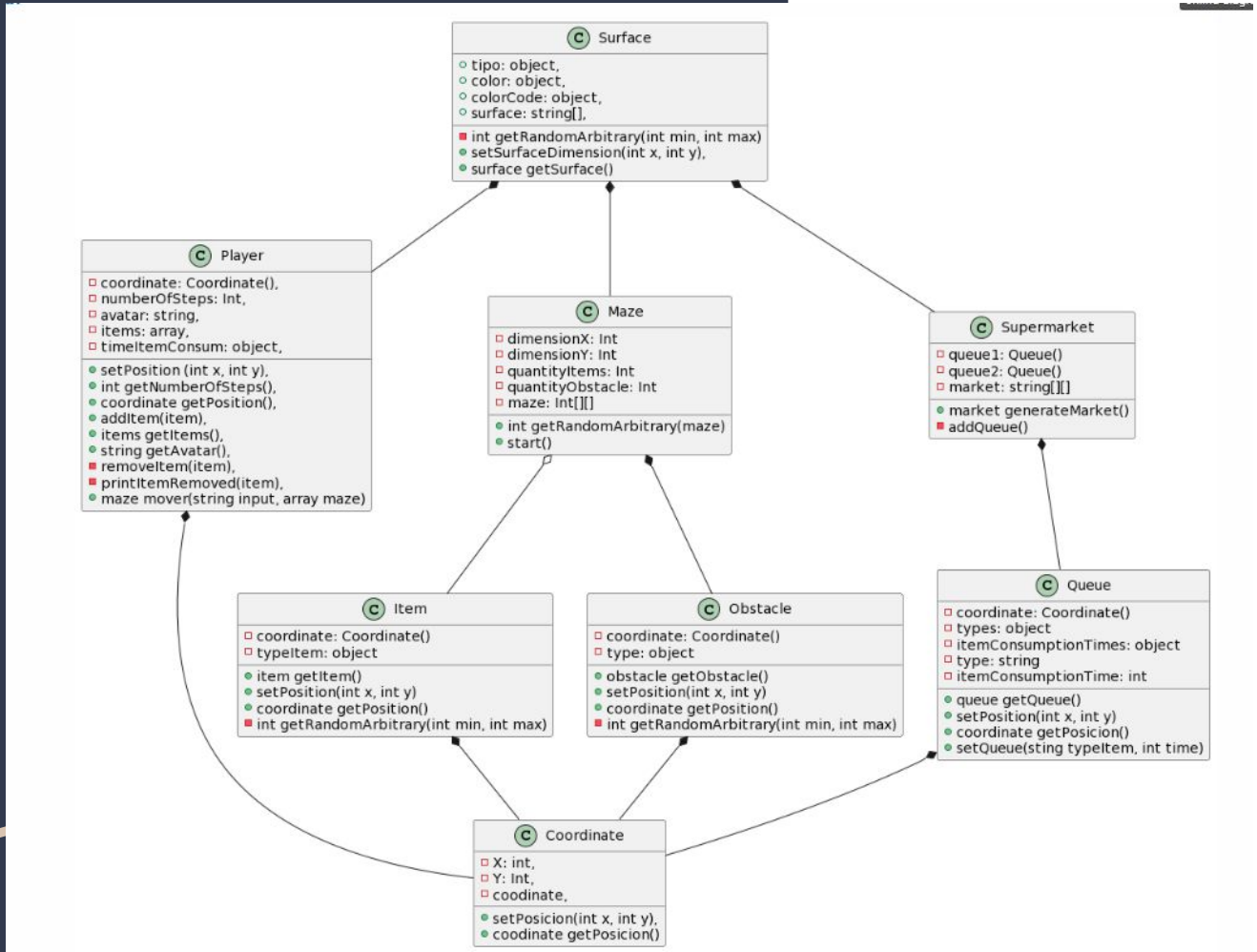

Surface
tipo: object,
color: object,
colorCode: object,
surface: string[],
int getRandomArbitrary(int min, int max)
setSurfaceDimension(int x, int y),
surface getSurface()
Player
coordinate: Coordinate(),
numberOfSteps: Int,
avatar: string,
items: array,
timeItemConsum: object,
setPosition (int x, int y),
int getNumberOfSteps(),
coordinate getPosition(),
addItem(item),
items getItems(),
string getAvatar(),
removeItem(item),
printItemRemoved(item),
maze mover(string input, array maze)
Maze
dimensionX: Int
dimensionY: Int
quantityItems: Int
quantityObstacle: Int
maze: Int[][]
int getRandomArbitrary(maze)
start()
Supermarket
queue1: Queue()
queue2: Queue()
market: string[][]
market generateMarket()
addQueue()
Item
coordinate: Coordinate()
typeItem: object
item getItem()
setPosition(int x, int y)
coordinate getPosition()
int getRandomArbitrary(int min, int max)
Obstacle
coordinate: Coordinate()
type: object
obstacle getObstacle()
setPosition(int x, int y)
coordinate getPosition()
int getRandomArbitrary(int min, int max)
Queue
coordinate: Coordinate()
types: object
itemConsumptionTimes: object
type: string
itemConsumptionTime: int
queue getQueue()
setPosition(int x, int y)
coordinate getPosicion()
setQueue(sting typeItem, int time)
Coordinate
X: int,
Y: Int,
coodinate,
setPosicion(int x, int y),
coodinate getPosicion()

Diagrama de Flujo

Diagrama de Movimiento

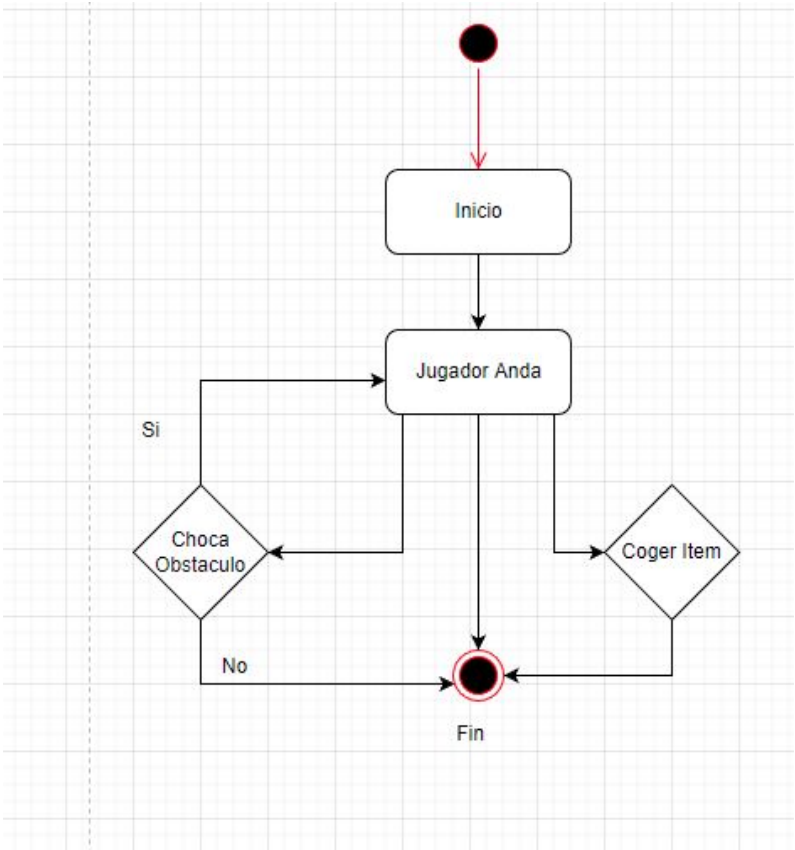


Diagrama de Flujo

Diagrama de Colas

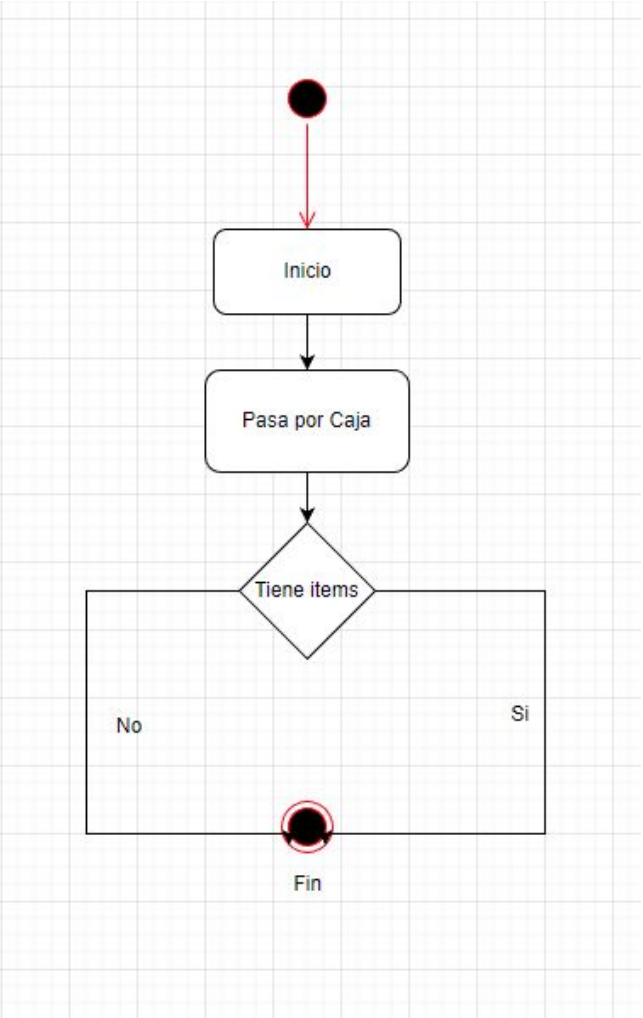


Diagrama de Flujo

Diagrama Completo

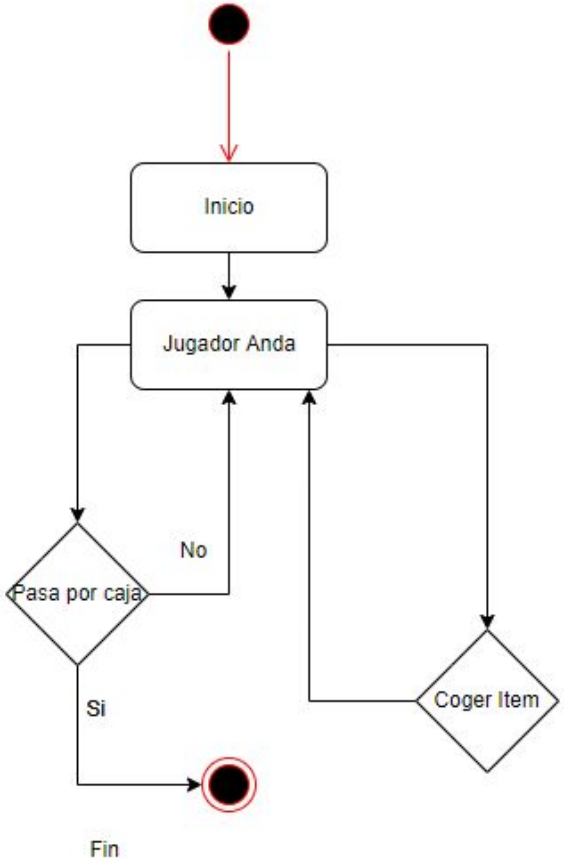


Diagrama de Objetos

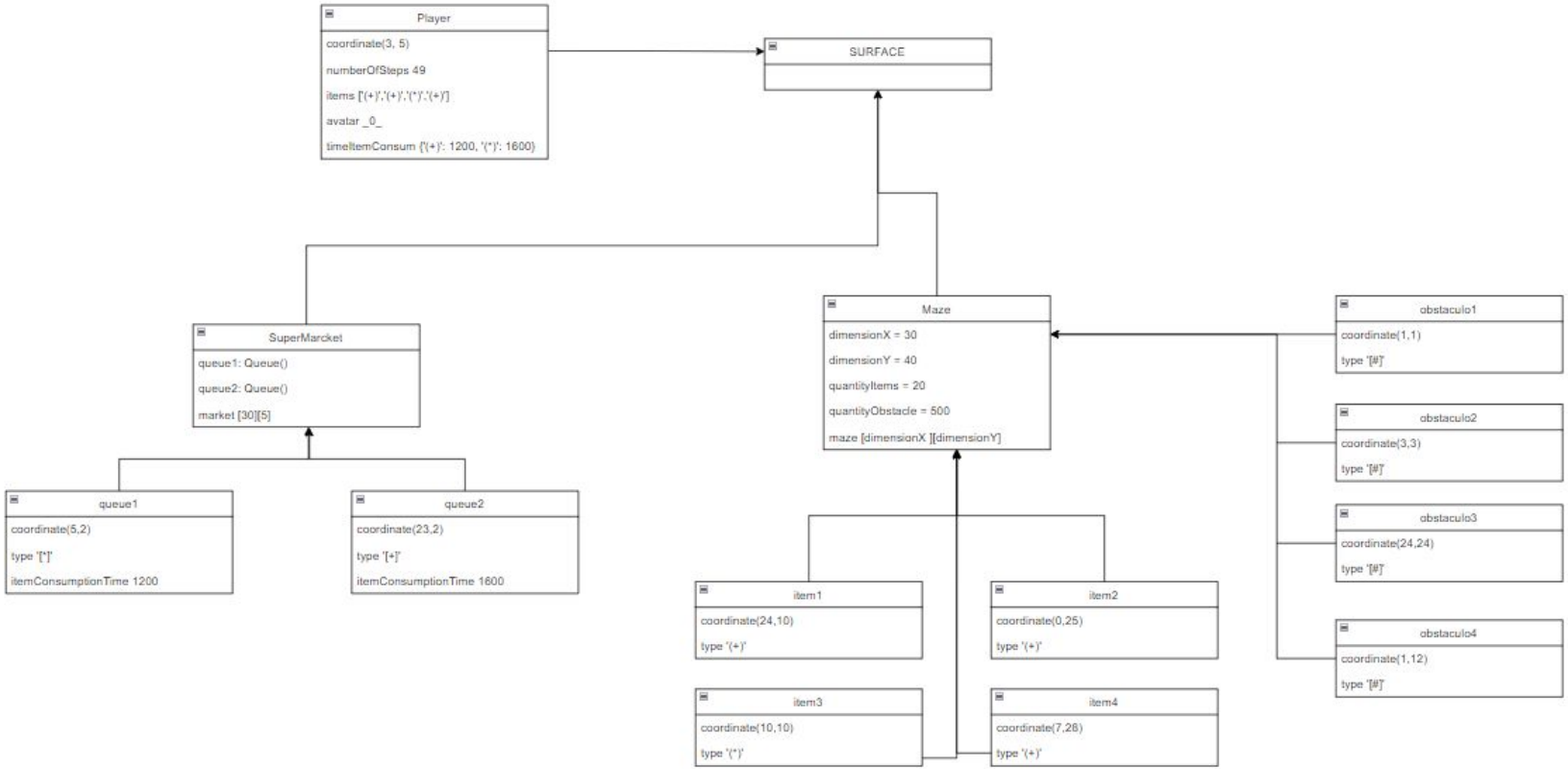
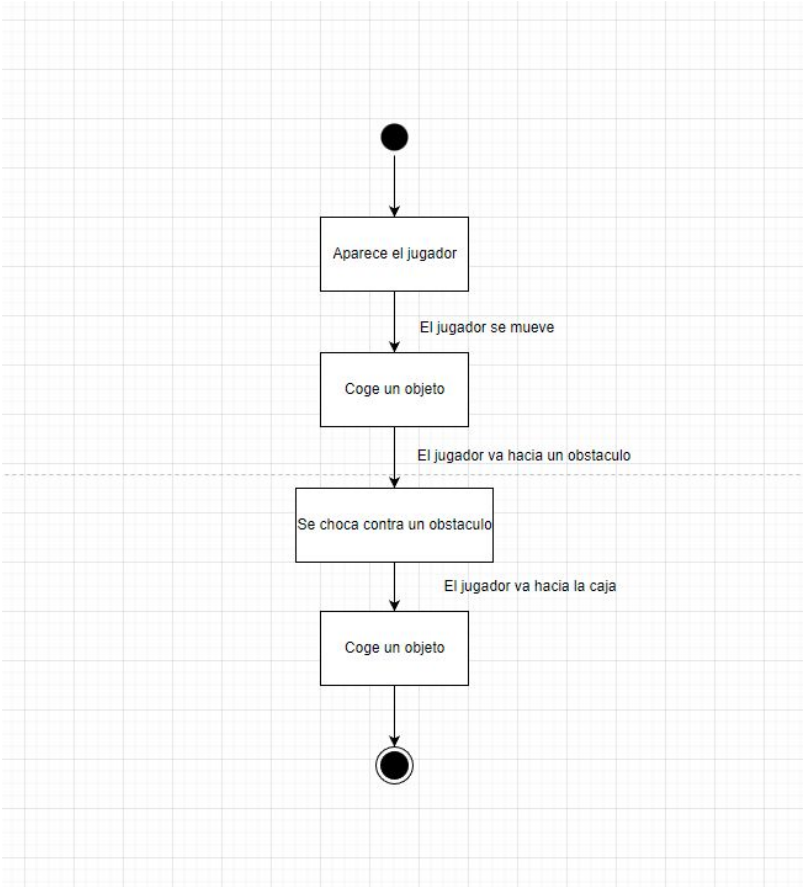
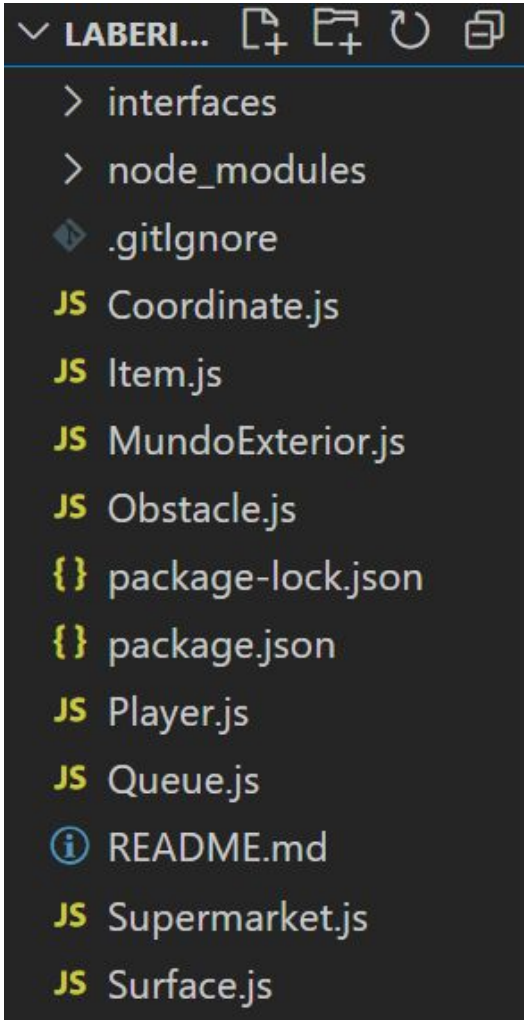

Player
coordinate(3, 5)
numberOfSteps 49
items ['(+)','(+)','(+)','(*)','(+)']
avatar _0_
timeItemConsum {'(+)': 1200, '(*)': 1600}
SURFACE
SuperMarcket
queue1: Queue()
queue2: Queue()
market [30][5]
Maze
dimensionX = 30
dimensionY = 40
quantityItems = 20
quantityObstacle = 500
maze [dimensionX][dimensionY]
obstaculo1
coordinate(1,1)
type '[#]'
obstaculo2
coordinate(3,3)
type '[#]'
obstaculo3
coordinate(24,24)
type '[#]'
obstaculo4
coordinate(1,12)
type '[#]'
queue1
coordinate(5,2)
type '[*]'
itemConsumptionTime 1200
queue2
coordinate(23,2)
type '[+]'
itemConsumptionTime 1600
item1
coordinate(24,10)
type '(+)'
item2
coordinate(0,25)
type '(+)'
item3
coordinate(10,10)
type '(*)'
item4
coordinate(7,28)
type '(+)'

Diagrama de estados



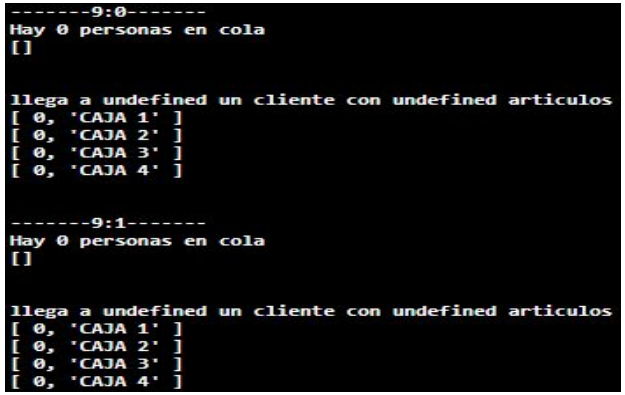
Estructura de carpetas



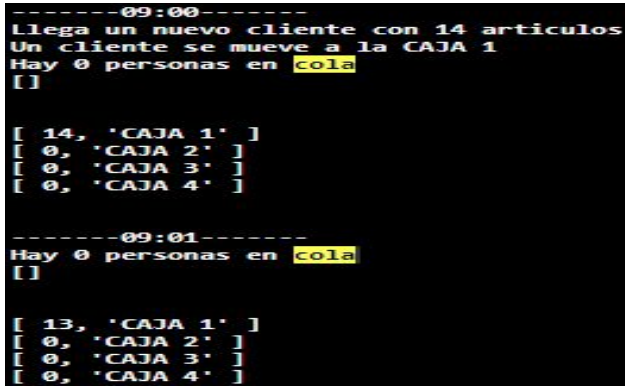
Smell Code(Colas)

Fallo en ítems vendidos: En la primera versión el programa fallaba en determinar la cantidad de artículos que entraban en la caja:

Antes



Después



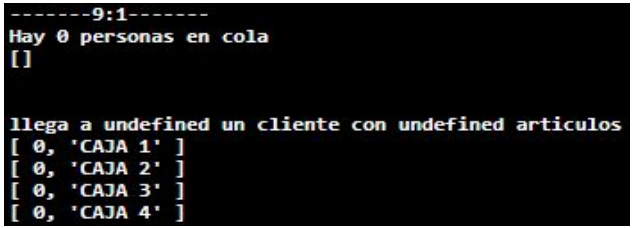
Smell Code(Colas)

Clase larga: Al tener una única clase todo el código y por lo tanto todas las acciones las realiza la misma clase :

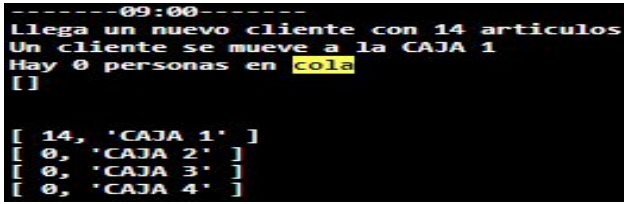
Smell Code(Colas)

Fallo en la hora: En la primera versión la hora comenzaba un minuto adelantada:

Antes



Después



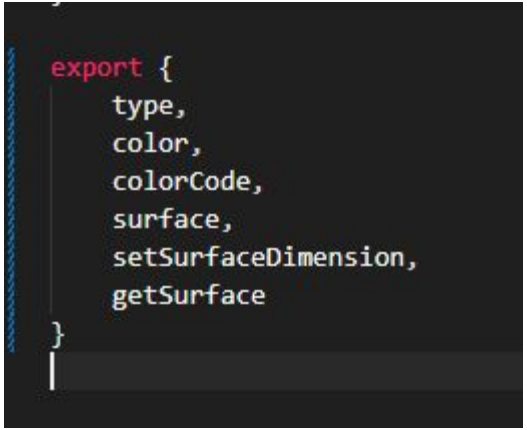
Smell Code

Demasiadas condiciones anidadas

```
if (option === 'W') {
    valueInt = positiony
    if (valueInt > 0) {
        const result = valueInt - 1
        if (maze[result][positionx] !== '[#]'){
            if(maze[result][positionx] === '(+)' || maze[result][positionx] === '(*)'){
                addItem(maze[result][positionx])
                maze[result][positionx] = '   '
            }
            if (maze[positiony][result] === '[*]'){
                await removeItem('(*)')
            }
            if (maze[positiony][result] === '[+]'){
                await removeItem('(+)')
            }
            setPosition (positionx, result)
        }
    }
}
if (option === 'S') {
    valueInt = positiony
    if (valueInt < maze.length) {
        const result = valueInt + 1
        if(maze[result][positionx] !== '[#]'){
            if(maze[result][positionx] === '(+)' || maze[result][positionx] === '(*)'){
                addItem(maze[result][positionx])
                maze[result][positionx] = '   '
            }
            if (maze[positiony][result] === '[*]'){
                await  removeItem('(*)')
            }
            if (maze[positiony][result] === '[+]'){
                await removeItem('(+)')
            }
            setPosition (positionx,result)
        }
    }
}
```

Smell Code

Se exponen todos los métodos y atributos de la clase



Smell Code

Depende de demasiadas clases

Smell Code

Demasiada complejidad en un solo punto. Usar un factory para generar información..

```
for (let index = 0; index < quantityObstacle; index++) {
    obstacle.setPosition(getRandomArbitrary(0, dimensionY - 1), getRandomArbitrary(0, dimensionX - 1))
    newObstacle = obstacle.getObstacle()
    obstacles.push(JSON.parse(JSON.stringify(newObstacle)))
}

for (let index = 0; index < quantityItems; index++) {
    item.setPosition(getRandomArbitrary(0, dimensionY - 1), getRandomArbitrary(0, dimensionX - 1))
    newItem = item.getItem()
    items.push(JSON.parse(JSON.stringify(newItem)))
}

for (let index = 0; index < market.length; index++) {
    const element = market[index]
    maze.push(element)
}

let obst, it

for (let index = 0; index <= obstacles.length - 1; index++) {
    obst = obstacles[index]
    maze[obst.posicion.y][obst.posicion.x] = obst.type
}
for (let index = 0; index <= items.length - 1; index++) {
    it = items[index]
    maze[it.posicion.y][it.posicion.x] = it.typeItem
}

async function start(maze) {
```