Memoria De la AA2

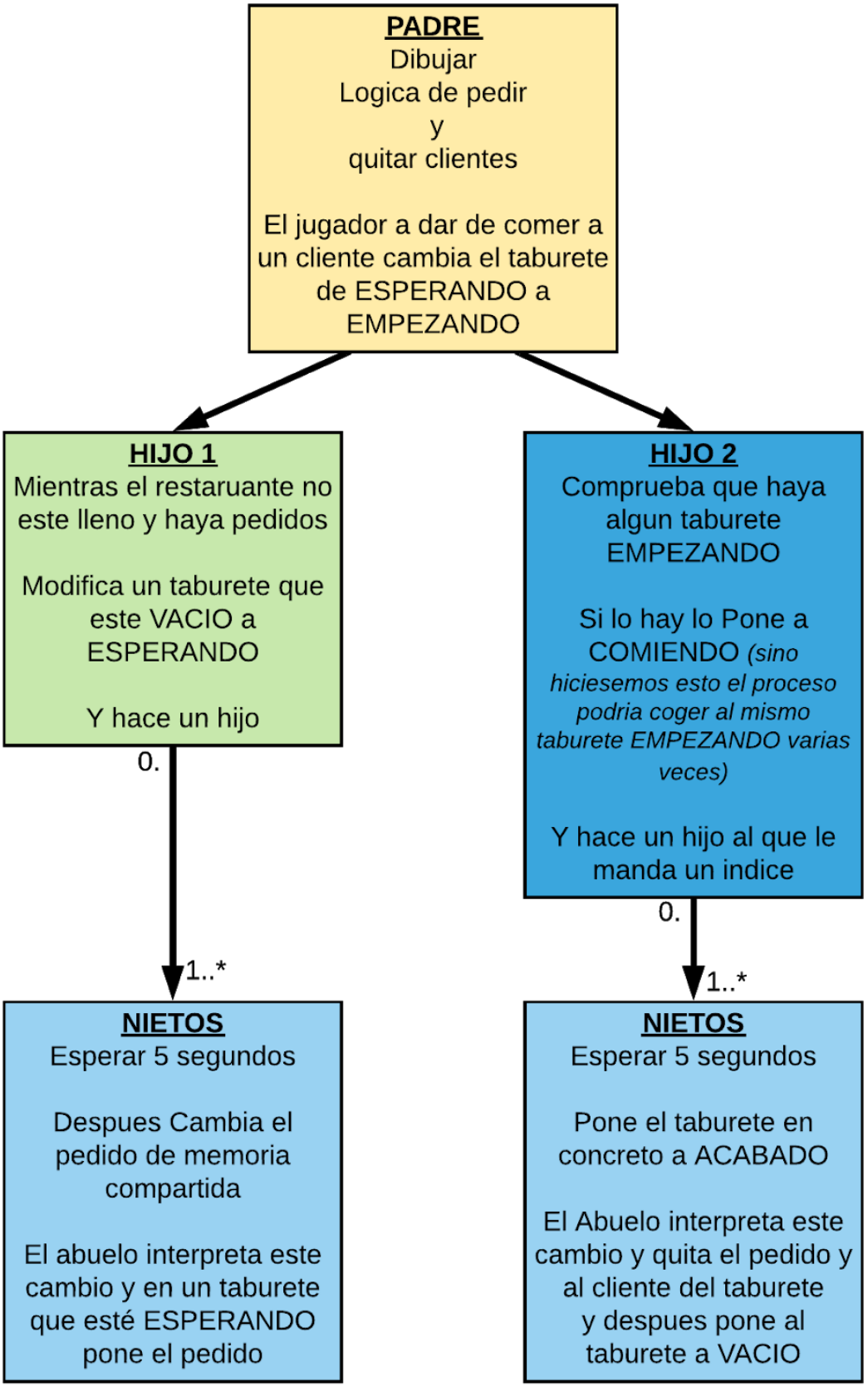
David Auladell
Maurizio Carlotta
Abraham Armas Cordero

2º CDI(dev) Tarda

ESCOLA DE NOVES TECNOLOGIES INTERACTIVES

centre adscrit a: UNIVERSITAT DE BARCELONA

Jerarquía de procesos



Memoria Compartida

¿Que hay?

Reservamos memoria para un struct Data**(253)** que contiene:

- Un array de tipo char de 3 posiciones (ocupan menos que un integer y pueden tener más estados que un bool)
- Un Color de SFML para definir de cual será el color del pedido

¿Quien la usa? y ¿Para que la Usa?

Antes de empezar los números en negrita y entre paréntesis son la línea de código en el main.cpp donde se encuentra todo esto

- El Padre:
    - Usa **el array** para determinar cuando el jugador le puede dar de comer a un cliente**(400)** y poner ese cliente a EMPEZANDO**(402)**, para saber cuando tiene que quitar un cliente**(419)** y ponerlo a VACÍO**(425)**.
    - Usa **el color** para saber si tiene que poner a un cliente**(437)** y rellenar un asiento que esté Esperando**(439)**
- Hijo 1:
    - Usa **el array** para determinar si hay sitio en el restaurante**(299)**, poner los hijos a Esperar**(301)**
    - Usa **el color** para dar la señal al padre de que hay un nuevo cliente**(307)**, y para decir de qué color es el pedido**(307)**.
- Hijo 2:
    - Usa **el Array** para saber si tiene que poner a comer a algún taburete**(323)**, para saber si tienen que comer**(504)** y ponerlos a comer**(507**, para hacerlo terminar de comer**(511)** y para avisar al padre de que ha acabado de comer y vacíe el taburete**(511)**

Semáforos

¿Como?, ¿Cuantos? y ¿Que protegen?

Los hacemos en el Main() y pasamos el ID por parametro a las funciones que lo necesiten, solo hemos usado un semáforo ya que protegemos la parte de los setters y getters que accede al atributo en concreto de la memoria compartida y como son instrucciones realmente rápidas no vemos necesario reservar más memoria para esto.

Waits y Signals

Procedo a poner la línea donde se encuentran los waits y signals de cada función, todas estas funciones son de *"struct Data" y los "o"* significan que si hace el signal de una línea no hace el de la otra.

- RestauranteLleno() Wait**(72)** Signal**(77)** o **(82)**
- AlguienEmpezando() Wait**(93)** Signal**(98)** o **(103)**
- AlguienSatisfecho() Wait**(113)** Signal**(118)** o **(123)**
- SetNewClientColor() Wait**(132)** Signal**(140)**
- SetTaburetes() Wait**(147)** Signal**(153)**
- GetNewClientColor() Wait**(162)** Signal**(167)**
- GetTaburetesState() Wait**(175)** Signal**(180)**
- GetTabureteVacio() Wait**(190)** Signal**(195)** o **(199)**
- GetClienteSatisfecho() Wait**(209)** Signal**(213)** o **(217)**

Conclusiones

- Hemos decidido hacerlo orientado a pulling entonces nadie conoce a nadie sino simplemente cambiando los estados de los recursos y comprobando esos estados continuamente saben lo que tienen que hacer.

  ¿Por qué? porque así nos forzamos a usar semáforos y a usarlos bien al final el objetivo de la práctica era aprender a proteger recursos si usamos signals es muy fácil hacer que ninguno lea o escriba mientras otro lo está haciendo, en cambio con pulling tienes que proteger bien la memoria porque se paralizan los procesos y no puedes controlar en qué orden usan la memoria compartida

- Podríamos haber hecho 2 semáforos, uno por cada recurso de la memoria compartida pero decidimos que es peor por:
  - Otro semáforo ocupa memoria que no hace falta
  - Empeora la legibilidad del código
  - Son tan breves las pausas que no necesitamos otro

- Hemos aprendido a usar semáforos, memoria compartida, hacer control de errores de estas dos cosas, a usar union.array junto al SETALL aunque no lo usaremos al final.