Practical Assignment 1

Borraz, Jordi
Muns, Nil
Ruiz, Ricard
Velasco, Blanca
Vergel, Héctor

Índice

Descripción de la escena

La escena del juego es un restaurante en el que hay dos zonas diferentes: la primera zona es el comedor, donde los clientes que llegan se sientan en las mesas y piden comida y la segunda zona es la cocina, donde el cocinero limpia los platos y prepara la comida. En el juego hay un total de 4 personajes o entidades diferentes, que son los siguientes:

- **El trabajador a tiempo completo**: esta entidad es el protagonista de la escena, se encarga de tomar nota a los clientes, cocinar la comida y llevarla a la mesa. Aparte, también limpia los platos y se encargará de exterminar a las hormigas en situaciones concretas.

- **El cliente**: esta entidad se dedica a buscar sitio en el comedor, si hay sitio disponible para sentarse se sentará y esperará a que llegue el camarero para tomarle la nota. Si el camarero tarda mucho en ir, este se enfadará y se irá del restaurante. En caso de que el camarero le traiga el pedido, se comerá la comida y, al irse, dejará el dinero encima de la mesa. En caso de que el cliente vea una hormiga, este se enfadará y se irá.

- **El limpia platos**: esta mejora estará disponible cuando el jugador haya conseguido el dinero suficiente para poder pagarla. Básicamente el limpia platos es una entidad que irá a buscar los platos sucios, los limpiará en el fregadero y los dejará disponibles para que el cocinero los pueda coger para preparar la comida.

- **Las hormigas**: estas entidades salen de un agujero que hay en el restaurante y su único objetivo es ir allí donde haya platos sucios. El jugador puede tapar el agujero de donde salen las hormigas si tiene suficiente dinero.

En este juego, al tratarse de un simulador de restaurante, hemos hecho que la mecánica principal sea la de gestión de recursos. Para lograr esto hemos implementado varias mejoras que el jugador puede ir desbloqueando a medida que consigue el recurso principal del juego, que es el dinero. El jugador solo puede interactuar con la escena dándole a los botones disponibles para conseguir las mejoras. El objetivo principal del juego es conseguir todo el dinero posible, para lo que el jugador debe atender rápidamente a los clientes y evitar que las hormigas lleguen a molestar.

Extras implementados

Hemos implementado todos los puntos adicionales:

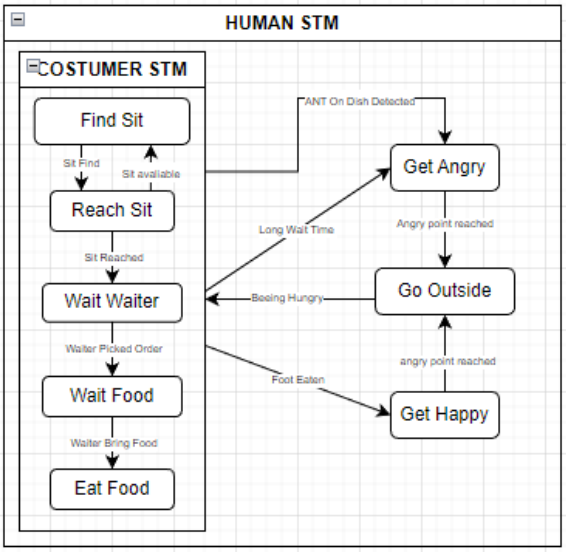
En primer lugar, tenemos el extra del personaje que tiene que implementar un "**Flock**" de más de 30 unidades. Hemos creado a la familia de hormigas, las cuales van en grupo buscando un "**Surrogate Target**".

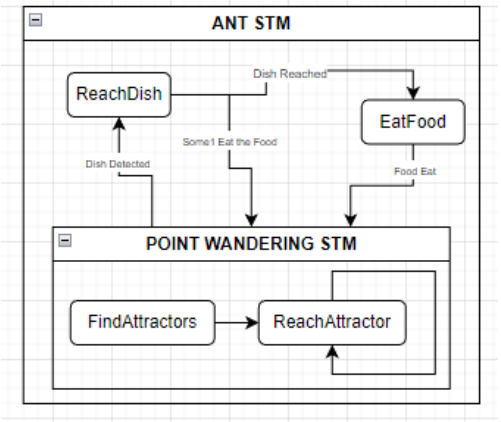
Respecto al personaje que tiene que tener 3 o más niveles de profundidad dentro de la máquina de estados tenemos al trabajador a tiempo completo, que cumple con todos estos requisitos, ya que dispone un total de 4 estados diferentes: "**Waiter Worker**, **Ant Killer**, **Wander Around** y **Pick Up Money"**.

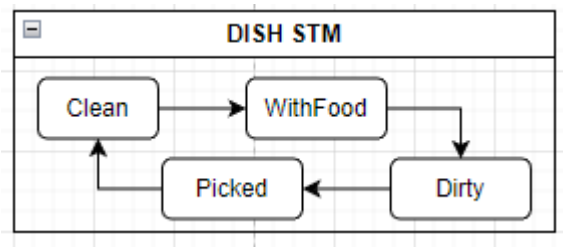
En el punto extra de interacción con la escena, el jugador puede conseguir varias mejoras: puede comprar un plato nuevo, adquirir un trabajador que se encargue de limpiar los platos y tapar el agujero por donde salen las hormigas. La interacción se hace presionando los botones que salen en la pantalla a modo de HUD.

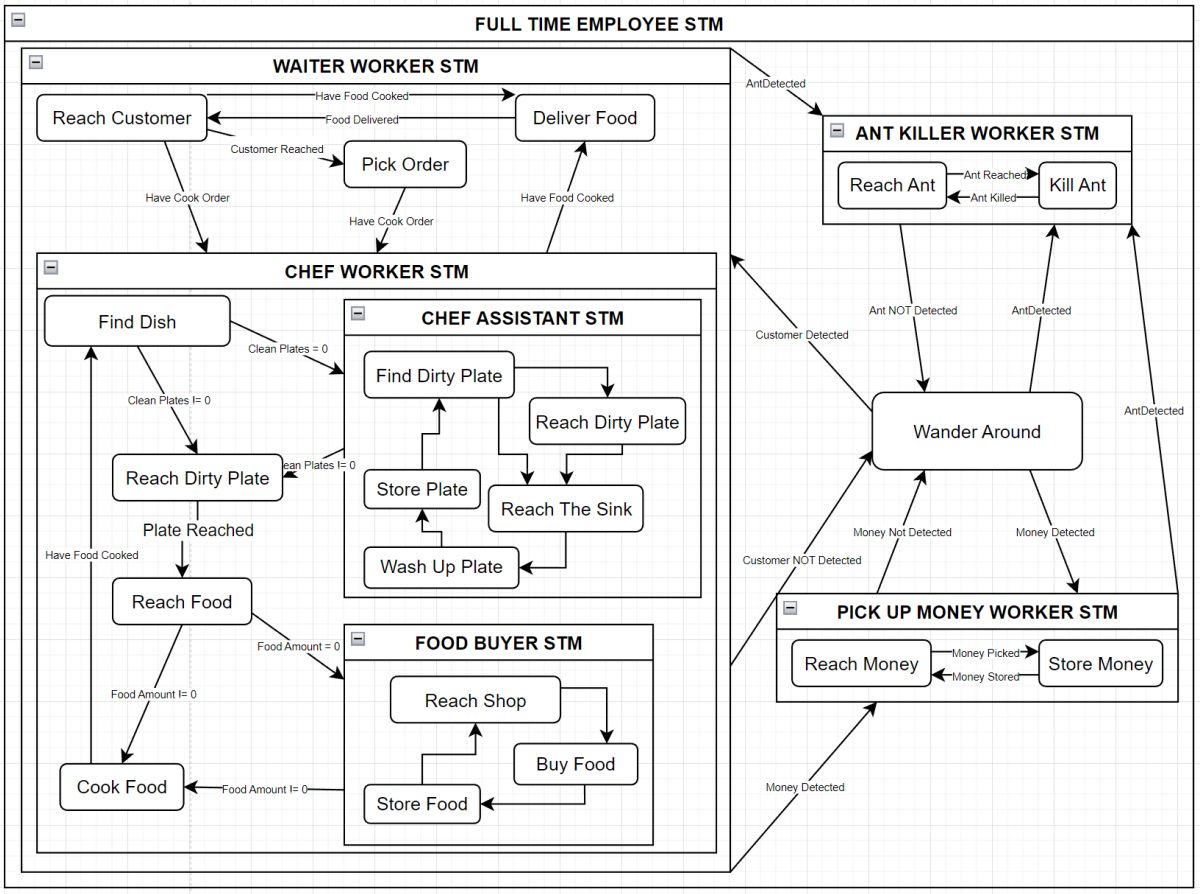
Para el último punto extra, el juego tiene un punto de superación. Cuando comienza el juego el trabajador solo tiene dos platos, lo que hace que siempre tenga que estar limpiándolos y yéndolos a buscar todo el rato porque son los únicos que tienes. Esto hace que el jugador pierda mucho tiempo y que muchas veces no pueda llegar a tomar nota a los clientes y estos se vayan. A medida que consigues platos avanzas más rápido, pero el hecho de que los tengas que ir a buscar mesa por mesa también hace que no seas lo más productivo posible. Es por esto que la mejora del limpia platos te soluciona ese problema, ya que esta entidad irá a buscarlo a las mesas y los lavará. Aparte de estas dos mecánicas, la única manera realmente efectiva que se tiene para parar a las hormigas es tener el suficiente dinero para tapar el agujero de donde salen. Este hecho hará que el jugador tenga esas ganas de conseguir dinero y conseguir mejorar su restaurante.

Diagramas de todas las FSM









Especificación de la FSM del “Trabajador a tiempo completo”

La máquina de estados más extensa que tenemos es la que usa el trabajador a tiempo completo, este empieza siempre en “Wander Around”, a partir de ahí puede pasar a los siguientes estados:

Si el trabajador detecta que un cliente pasa dentro de la sub-máquina de estados del *WAITER WORKER*, esta tiene 3 estados y 5 transiciones que son las siguientes:

- **State reachCostumer:** al entrar pone como *target* al cliente e invoca el *arrive* para que el chef se mueva a la ubicación de este. Este estado solo sirve para mover el *player* de posición, o sea que en el *in-state* no hace nada y el *exit* lo usa para indicar que ya ha llegado al *target*.

- **State pickOrder**: en el *enter* pone el tiempo a cero y pone la variable *bool* de *activeOrder* en *false*. En el *in-state* se suma la variable *elapsedTime* para comprobar en su transición si supera el tiempo deseado y entonces en el *exit* establece la variable *activeOrder* en *true*.

- **State deliverFood**: se activa el *arrive* y se marca el *target* de nuevo en el *enter* para que el chef vaya hacia el cliente, en el *in-state* se suma el tiempo y en el *exit-state* se pone el *arrive* en *false*, se pone el *transform* del plato como *null* para que ya no esté anclado al player, se borra el cliente de la lista y también se marca la variable de la *activeOrder* en *false*.

- **Transition customerReached**: la transición tiene como *trigger* la función *SensingUtils.DistanceTotarget* para ver si el *gameObject* en cuestión ha llegado al cliente. Para hacer la acción de transición pasa del estado *reachCustomer* a *pick Order* si se cumple el *trigger*.

- **Transition haveOrder**: como *trigger* esta transición se centra en mirar si ha pasado un tiempo determinado, que es el tiempo que la entidad necesita para tomar la orden. Pasará del estado *pickOrder* al estado de chef.

- **Transition haveCooked**: para activar la transición se necesita que la variable *haveCookedFood* se encuentre en *true*, pasará del estado *chef* al estado *deliverFood*.

- **Transition FoodDeliver**: el *trigger* de esta transición necesita ver si se ha llegado al cliente, en caso de que esto ocurra se pasará del estado *deliverFood* al de *reachCustomer*.

- **Transition angryCustomer**: básicamente esta transición mira si hay algún *tag* del *customer* diferente para detectar si hay algún cliente que se ha cabreado, si pasa esto igualmente se pasará del estado *deliverFood* al *reachCustomer*.

Una vez el trabajador tenga un pedido pasará a la siguiente sub-máquina de estados, que en este caso es la del *CHEF WORKER*. Esta tiene un total de 4 estados y 6 transiciones:

- **State findDish**: este estado es un tanto especial, ya que no hace nada, simplemente es un estado que espera a que alguna transición lo saque del propio estado. Este será el estado inicial de esta máquina de estados.

- **State reachPlate**: al entrar activa el método *arrive* y pone como *target* al plato para hacer que el player se dirija para allá, no tiene nada en el *in-state*, ya que el estado sirve para *setear* que tiene que ir al plato. En el *exit* desactiva el método *arrive* y activa un método específico llamado *PlaceOn* para cambiar el *transform* del plato.

- **State reachFood**: en el *enter-state* activa el método *arrive* y pone como *target* a la nevera para hacer la acción de que el cocinero va a buscar la comida a la nevera. Por último, en el *exit* vuelve a desactivar el *arrive*.

- **State cookFood**: al entrar en el estado la variable *bool* de *cooking* se activa, se pone a cero la variable de *elapsedTime* para poder calcular bien en el *in-state* cuánto tiempo ha de estar cocinando y se activa el método *arrive* y se pone como *target theCook*. En el *in-state* se hace una condición usando un *if* para ver si el chef está en el radio de la cocina y se va sumando el tiempo para así calcular el tiempo que ha de estar dentro. Finalmente, en el *exit-state* se ponen las variables *cooking* y *arrive* en *false*, la variable *haveCookedFood* en *true* y se usa el método *PlaceFood* para volver a coger el plato.

- **Transition havePlates**: la transición busca un plato limpio y, si lo encuentra, entonces automáticamente sale de la transición, pasaría del estado *findDish* al *reachPlate*.

- **Transition haveNoPlates**: esta transición hace exactamente igual que lo que hacía la anterior, pero al revés: busca platos con el *tag DISH_DIRTY* y, si los encuentra, automáticamente sale de la transición. Pasaría del estado *findDish* al *chefAssistant*.

- **Transition plateReached**: mira si la distancia al *target* es la necesaria para poder cogerlo y, si puede, pasa del estado *reachPlate* al de buscar comida a la nevera.

- **Transition haveFood**: para activar esta transición, el código necesita tener varios parámetros: el primero es ver si en la variable de *totalFood* al menos hay más de cero, si esto pasa entonces el *player* también deberá de estar a una distancia determinada de la nevera o también si ya está cocinando podrá efectuar la transición. Esta transición puede pasar en dos ocasiones: cuando el jugador está en el estado de *foodBuyer* que pasará al estado de *cookFood* o cuando ya ha llegado a la comida y está en el estado *reachFood,* el cual también pasará al estado de cocinar.

- **Transition haveNoFood**: en caso en el que el chef vaya a la nevera y no tenga comida, esta transición se activará, se mirará el valor de la variable *totalFood*. En este caso se pasaría del estado *reachFood* al de *foodBuyer*.

- **Transition FoodCooked**: el *trigger* se activará cuando el tiempo transcurrido sea mayor que el tiempo necesario para cocinar la comida. Esto se sabrá usando la variable del *blackboard cookFoodTime*. Se pasará del estado *cookFood* al de *findDish*.

En caso de que el chef tenga que ir a comprar comida, se usará otra sub-máquina llamada *FOOD BUYER*, esta solo se llamará si el jugador no tiene comida, o sea, si el valor de *foodAmount* es inferior a uno. Esta FSM tiene un total de 3 estados y 3 transiciones:

- **State *reachTheCupboard***: en este estado cuando entra activa el método *arrive* y le pone al *target* de este el *Cupboard* (este *Cupboard* más que nada es el nombre que le hemos puesto al sitio donde el cocinero va a comprar los alimentos si es que no tiene en la nevera). En el *in-state* no hace nada y en el *exit-state* desactiva el método *arrive*. Este es el estado inicial de la máquina de estados.

- **State pickFood**: este estado sirve para hacer que el cocinero se pare delante del *CupBoard* unos instantes para simular que este está comprando. En el *in-state* se pone la variable *bool* de *buying* en *true* y se pone el tiempo transcurrido en cero. Ya dentro

del estado se activa el tiempo transcurrido usando el *deltaTime*. El *exit* queda vacío en este estado en concreto.

- **State storeFoodOnFridge**: se activa el *arrive* y se pone como *target* a la nevera en el *in-state*, a continuación en el *enter* se activa la variable *elapsedTime* para calcular el tiempo y finalmente en el *exit* se desactivan tanto la variable de *buying* y la del *arrive* y se ejecuta el método del *blackboard* llamado *storeFood*.

- **Transition reachedTheCupboard**: la *trigger condition* de esta transición es ver si la variable *buying* es verdadera o si mirar, usando el método *distanceToTarget,* si el chef está lo suficientemente cerca del *cupboard*. Esta transición hará que se pase del estado *reachTheCupboard* al de *pickFood*.

- **Transition foodPicked**: mira si el tiempo transcurrido es mayor o igual al tiempo que tarda el chef en comprar la comida y acepta la condición. También puede ser que el código detecta que la variable de comprar *buying* sea *true* y ya de por válida la transición. La transición hará que se pase del estado *pickFood* al de *StoreFoodOnFridge*.

- **Transition reachedTheFridge**: esta transición simplemente mira si el *player* está lo suficiente cerca de la nevera y, si lo está, pasa del estado *storeFoodOnFridge* al de *reachTheCupboard*.

Cuando el trabajador tiene que encontrar platos y todos los que hay en la escena están sucios, se entrará de nuevo dentro de otra sub-máquina de estados, la cual se encargará de limpiar los platos para que el chef pueda poner la comida en ellos. Esta máquina de estados se llama *CHEF ASSISTANT STM*, tiene un total de 5 estados y 7 transiciones:

- **State findDirtyPlate:** este es el estado inicial de esta máquina de estados y no tiene nada dentro, simplemente sirve para que alguna transición lo saque del propio estado.

- **State reachDirtyPlate**: sirve para activar en el *enter-state* el *arrive* y ponerle como *target* el plato sucio en cuestión, en el *exit* el código busca el plato y, si lo encuentra, usa un método llamado *Pick* que ancla el plato al *transform* del chef.

- **State reachTheSink**: es un estado donde solo tiene código en el *enter* y simplemente se usa para marcar el *target*, en esta ocasión la pica (*theSink*) y activar el *arrive*.

- **State washUpPlate**: en el *in-state* se pone el tiempo transcurrido a cero, en el *enter-state* se va sumando el tiempo haciendo uso del *deltaTime* y el *exit* queda vacío.

- **State storePlate**: al inicio del estado se cambia el punto del *arrive* al de la pila de platos y, a continuación, en el *exit-state* se pone la variable *theDish* en *null*, ya que el chef ya se ha quedado sin platos y se usan los métodos *Wash* y *PlaceOn* para dejar el respectivo plato limpio y en la pila de platos.

- **Transition dirtyPlateDetected**: esta transición sirve para ver si en la escena hay algún plato el cual tenga como tag *DIRTY_PLATE*. Si es el caso se activará esta transición y se pasará del estado *findDirtyPlate* al *reachDirtyPlate*.

- **Transition AlreadyPickedADish**: mira si el valor de *theDish* es diferente a *null*, para darse cuenta si ya ha cogido un plato. Si se cumple la condición se pasará del estado *findDirtyPlate* a *reachTheSink*.

- **Transition some1WantToWashMyDish**: la transición busca si hay algún plato con el tag *DISH_DIRTY* y si no lo hay se efectúa la transición. Se pasa del estado *reachDirtyPlate* a *findDirtyPlate*.

- **Transition sinkReached**: básicamente usa la función *DistanceToTarget* para mirar si el *player* está lo suficiente cerca de la pica. Si lo está, pasará del estado de *reachTheSink* al de *washUpPlate*.

- **Transition dishReached**: hace lo mismo que la anterior función, pero en este caso mira si está lo suficientemente cerca requerido por el código para poder coger el plato (*theDish*) y pasará del estado *reachDirtyPlate* al *reachTheSink*.

- **Transition pileReached**: mira si el *player* está lo suficientemente cerca de la pila de platos y, si lo está, se cumplirá la condición. Esto hará que se pase de estado de *storePlate* al de *findDirtyPlate*.

- **Transition washedUpDish**: esta transición servirá para ver si el tiempo transcurrido es mayor que el tiempo requerido que el chef necesita para limpiar el plato. Una vez esto se cumpla se pasará del estado *washUpPlate* al de *storePlate*.

Ya fuera de la máquina de estados de *WAITER WORKER* que conglomeraba todas las sub-máquinas de estados explicadas anteriormente, tenemos a una sub-máquina llamada *ANT KILLER*. Esta máquina será llamada cuando el *player* detecte alguna hormiga. En total tiene 2 estados y 2 transiciones:

- **State reachAnt**: este estado es predefinido para la máquina de estados y sirve para indicar dónde debe de ir el *player* y dónde está la hormiga. En el *in-state* activa el método *arrive* y se pone como *target* a la hormiga que más cerca tenga el *player*, en el *enter-state* no se ejecuta nada y en el exit-state se pone en *false* el método *arrive*.

- **State killAnt**: en el *in-state* se resetea la variable *elapsedTime.* A continuación, en el *in-state* se procede a empezar a sumar el tiempo transcurrido en relación con el *deltaTime* y finalmente en el *exit-state* se ejecuta el método situado dentro del *blackboard* llamado *KillAnt*.

- **Transition antReached**: la transición mirará si el *player* en algún momento está lo suficiente cerca para darse cuenta de que tiene una hormiga cerca y, si ocurre, pasará del estado *reachAnt* al de *killAnt*.

- **Transition antKilled**: mirará si el tiempo transcurrido es superior al tiempo que el chef necesita para matar a la hormiga y, si eso ocurre, querrá decir que ya la ha eliminado de la escena y puede pasar del estado *killAnt* al de *reachAnt*.

La última FSM que tiene el trabajador es la de recoger el dinero. Esta se pondrá en marcha a medida que los clientes acaben de comer y dejen el dinero sobre las mesas. Esta sub-máquina se llama *PICK UP MONEY* y tiene un total de 2 estados y 2 transiciones:

- **State reachMoney**: este es el estado inicial de la máquina de estados. Sirve para darle la información al *in-state* de que se tiene que activar el *arrive* y poner el *target* al *gameObject* del dinero, en el *enter-state* no se hace nada y el *exit-state* se pone en *false* el método *arrive* y con el método *setParent* se ancla el *transform* del billete a la posición del chef.

- **State storeMoney**: en el *in-state* se activa de nuevo el *arrive*, pero en este caso se pone de *target* a la máquina registradora, en el *exit-state* se pone en *false* el *arrive* porque el *player* ya ha llegado y se procede activar el método *StoreMoney*.

- **Transition moneyPicked**: esta transición sirve para ver si el *player* está lo suficientemente cerca del dinero para poder cogerlo, si es así, se pasa del estado *reachMoney* al de *storeMoney*.

- **Transition cashierReached**: esta transición también detecta si el *player* ha llegado al radio de detección de la caja registradora y si es así pasa del estado de *storeMoney* al de *reachMoney*.