UNIVERSITAT JAUME • I

Development of non-linear and autonomous NPC

Francisco José Escriche Grau

Final Degree Work

Bachelor's Degree in
Video Game Design and Development

Universitat Jaume I

July 11, 2019

Supervised by: José Martínez Sotoca

Abstract

This document represents my Final Degree Work on the Bachelor's Degree in Video Game Design and Development.
This project tries to find one solution for the repetitive, predictable and non-human behavior that some games presents nowadays. For that, we have created a non-linear artificial intelligence that pursue his own goals on a marketplace environment.

CONTENTS

Figure Index:

.

1

CHAPTER

Introduction

Contents:

1.1 Motivation

The hardware and technologies improvements in the field of video games in the last two decades have caused a great leap in content quality and quantity. Many genres have been gradually changing, until little by little have been approaching to sandbox games [1] , requiring large amounts of content, mapping, missions, NPC (non playable characters) , etc. to be attractive to the general public.

Figure 1.1: Sandbox game concept are affecting a lot of game sagas and genres. Screenshot taken from Zelda Breath of the Wild.

However, not all aspects of the titles have evolved with the same "quality". AI (Artificial intelligence) has often been left in the background, becoming increasingly evident that it is necessary to provide innovations.

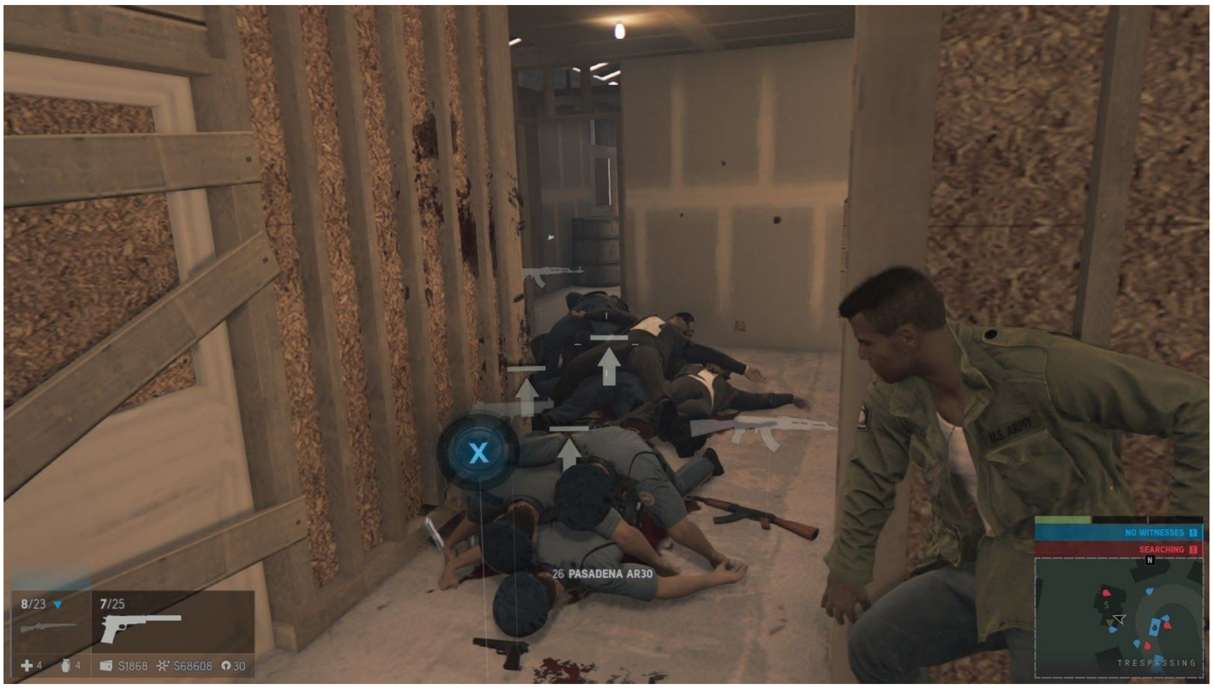


Figure 1.2: A clear example of that is Mafia 3, where a deficient AI and tons of "bugs" destroys the game experience [2][3].

In this final degree work, we propose a solution for the non-human, irrelevant and/or predictable behavior that many RPGs (role-play games) owns. We want characters with natural and not repetitive behaviours.

1.2 Objectives

With this motivation in mind, we consider making a complex logic, where the NPCs must take decisions depending on the rest of game elements.

To achieve that, we are going to recreate a small market, with 3 shops that will sell the same products, but they will have different prices and quality. In addition to that the shopman will have different attitudes towards the customers and the foes. The NPCs will buy products and will have multiple statistics such as relationship with the salesman, satisfaction with the purchased products and personality.

These statistics will make NPCs prefer to buy from a particular salesman. We should always get different results in each game execution in general terms, making a non linear AI in the sense that with the same event, AI will not react at the same way and this reaction will not be cyclic.

As a result of the NPC decisions, it is possible that one or two salesman go bankrupt. All the system will be AI driven.

It is important to remark that this project is not designed as a standalone game because it is designed to be a part or a system within a game, so the objectives are focused on this fact.

Therefore the objectives pursued are the following:

Objective 1:	Creation of an adaptable non-linear AI that pursues the satisfaction of its needs and objectives.
Objective 2:	Creating an AI that makes decisions adaptively according to the stimulus received.
Objective 3:	Ensure that each execution of the game do not happens the same.
Objective 4:	Possibility of designing new behaviors in the AI without making any hardly changes.
Objective 5:	Possibility of changing the behavior of the implemented behaviors without affecting the rest of the AI.
Objective 6:	Possibility of adding GOAP (Goal-Oriented Action Planning) in the future, when the NPC will be able to have different goals to accomplish at the same time and he will need a manager to establish priorities.

1.3 Environment

As a development environment, I choose Unity [4] (Unity version: 2018.3.9f1) as game engine and C# as the programming language.

The reasons for this choice are multiple:

- Multipurpose, versatile, and free game engine.
- I have more than 2 years of experience using it, so I have confidence, speed and knowledge of its use.
- The standard assets of Unity can be used for some specific things. For example, the standard character comes with the animations and a small movement script that allows to move it by script, simply by marking the target.

The rest of the models are made in Blender [5], also chosen for being versatile and free.

As well, this document was made following the LaTeX [6] recommendations to make the Final Degree report.

2

CHAPTER

Planning and resources

2.1 Planning

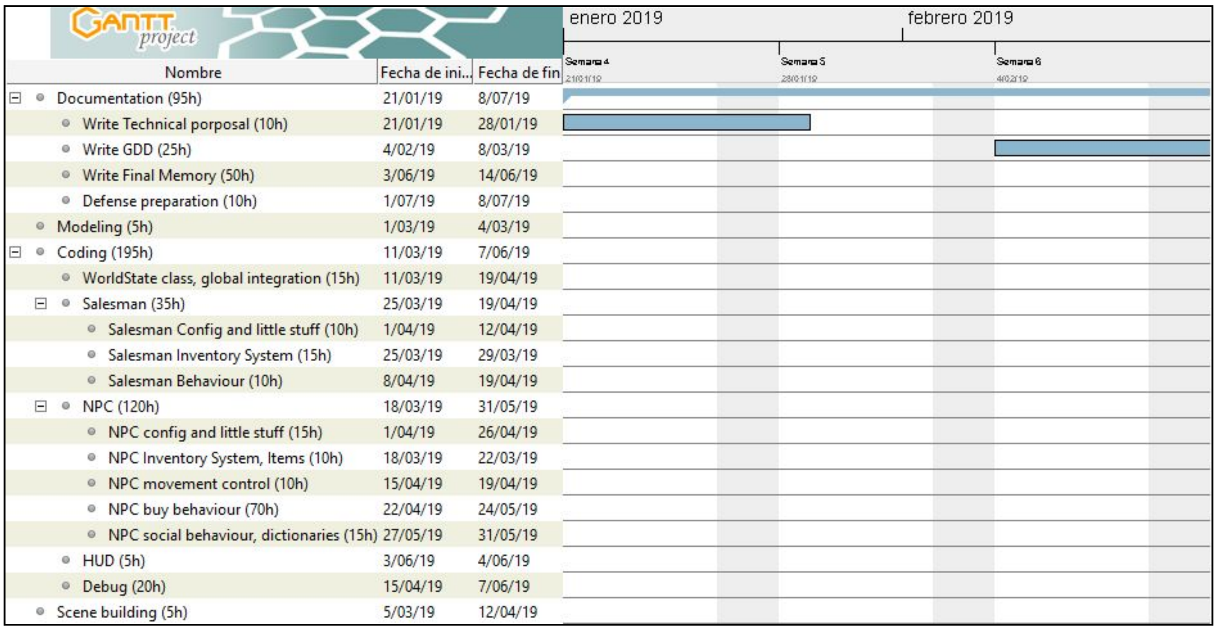
The final project planning is shown in Figure 2.1.1. [7] This planning has suffered several changes over time, depending on the needs and problems encountered. The most complex and relevant part is the NPC buy behaviour because it is basically the core of the project.

2.2 Planning evaluation, risk

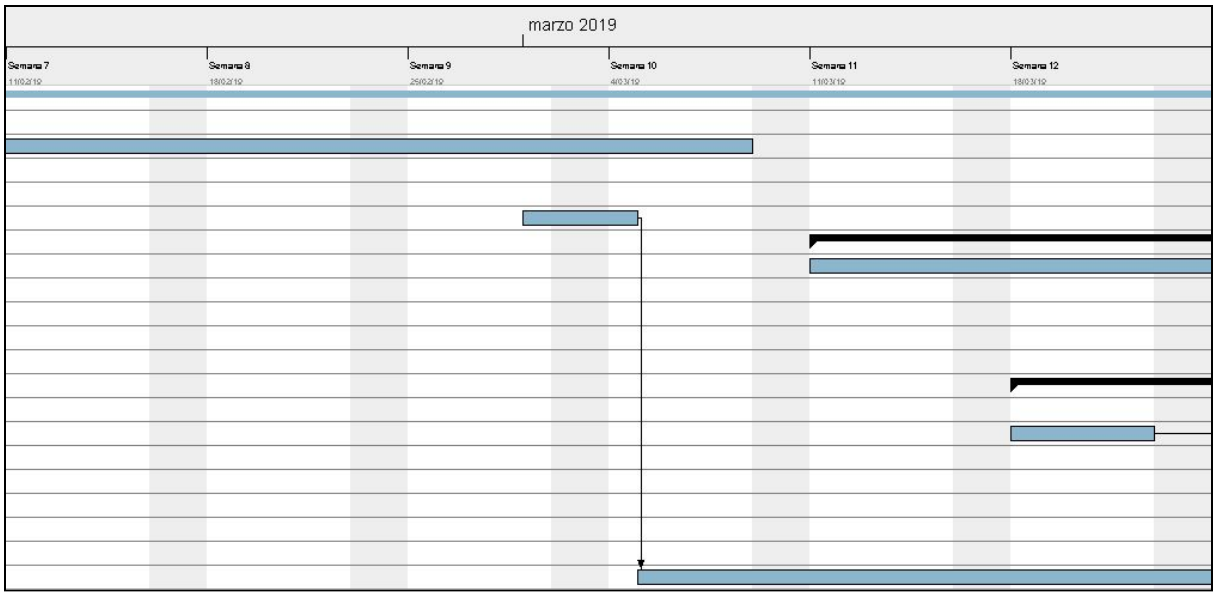
The project presents several risks, the first comes in the planning itself, since it is not possible to determine exactly how long it will take to develop all the mechanics proposed, especially as regards AI. I've worked with AI before, but always with state machines (complex, but state machines) and sensors, but never creating a completely autonomous AI that tries to pursue its own goals, and not behave equally to the same stimulus. This is a challenge, because it is hard to make an approximation without previous references.

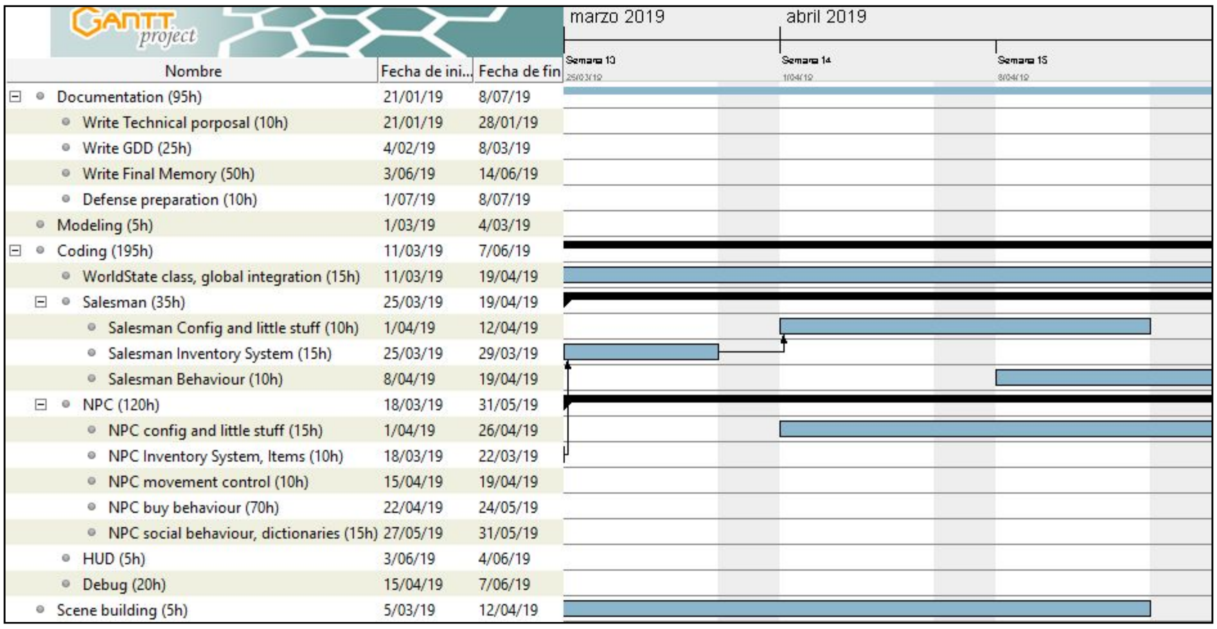
The second risk comes from the nature of the project, being an AI based project, the bugs are more difficult to fix and detect, since the combinations are larger, and even more if we remember that the AI is non-linear.

However there is an advantage in this project, it's developed in Unity. It is a game engine that I have already used a lot and I know all I can and can not do.



Nombre	Fecha de ini...	Fecha de fin
⊟ Documentation (95h)	21/01/19	8/07/19
Write Technical porposal (10h)	21/01/19	28/01/19
Write GDD (25h)	4/02/19	8/03/19
Write Final Memory (50h)	3/06/19	14/06/19
Defense preparation (10h)	1/07/19	8/07/19
Modeling (5h)	1/03/19	4/03/19
⊟ Coding (195h)	11/03/19	7/06/19
WorldState class, global integration (15h)	11/03/19	19/04/19
⊟ Salesman (35h)	25/03/19	19/04/19
Salesman Config and little stuff (10h)	1/04/19	12/04/19
Salesman Inventory System (15h)	25/03/19	29/03/19
Salesman Behaviour (10h)	8/04/19	19/04/19
⊟ NPC (120h)	18/03/19	31/05/19
NPC config and little stuff (15h)	1/04/19	26/04/19
NPC Inventory System, Items (10h)	18/03/19	22/03/19
NPC movement control (10h)	15/04/19	19/04/19
NPC buy behaviour (70h)	22/04/19	24/05/19
NPC social behaviour, dictionaries (15h)	27/05/19	31/05/19
HUD (5h)	3/06/19	4/06/19
Debug (20h)	15/04/19	7/06/19
Scene building (5h)	5/03/19	12/04/19





Nombre	Fecha de ini...	Fecha de fin
⊟ Documentation (95h)	21/01/19	8/07/19
Write Technical porposal (10h)	21/01/19	28/01/19
Write GDD (25h)	4/02/19	8/03/19
Write Final Memory (50h)	3/06/19	14/06/19
Defense preparation (10h)	1/07/19	8/07/19
Modeling (5h)	1/03/19	4/03/19
⊟ Coding (195h)	11/03/19	7/06/19
WorldState class, global integration (15h)	11/03/19	19/04/19
⊟ Salesman (35h)	25/03/19	19/04/19
Salesman Config and little stuff (10h)	1/04/19	12/04/19
Salesman Inventory System (15h)	25/03/19	29/03/19
Salesman Behaviour (10h)	8/04/19	19/04/19
⊟ NPC (120h)	18/03/19	31/05/19
NPC config and little stuff (15h)	1/04/19	26/04/19
NPC Inventory System, Items (10h)	18/03/19	22/03/19
NPC movement control (10h)	15/04/19	19/04/19
NPC buy behaviour (70h)	22/04/19	24/05/19
NPC social behaviour, dictionaries (15h)	27/05/19	31/05/19
HUD (5h)	3/06/19	4/06/19
Debug (20h)	15/04/19	7/06/19
Scene building (5h)	5/03/19	12/04/19

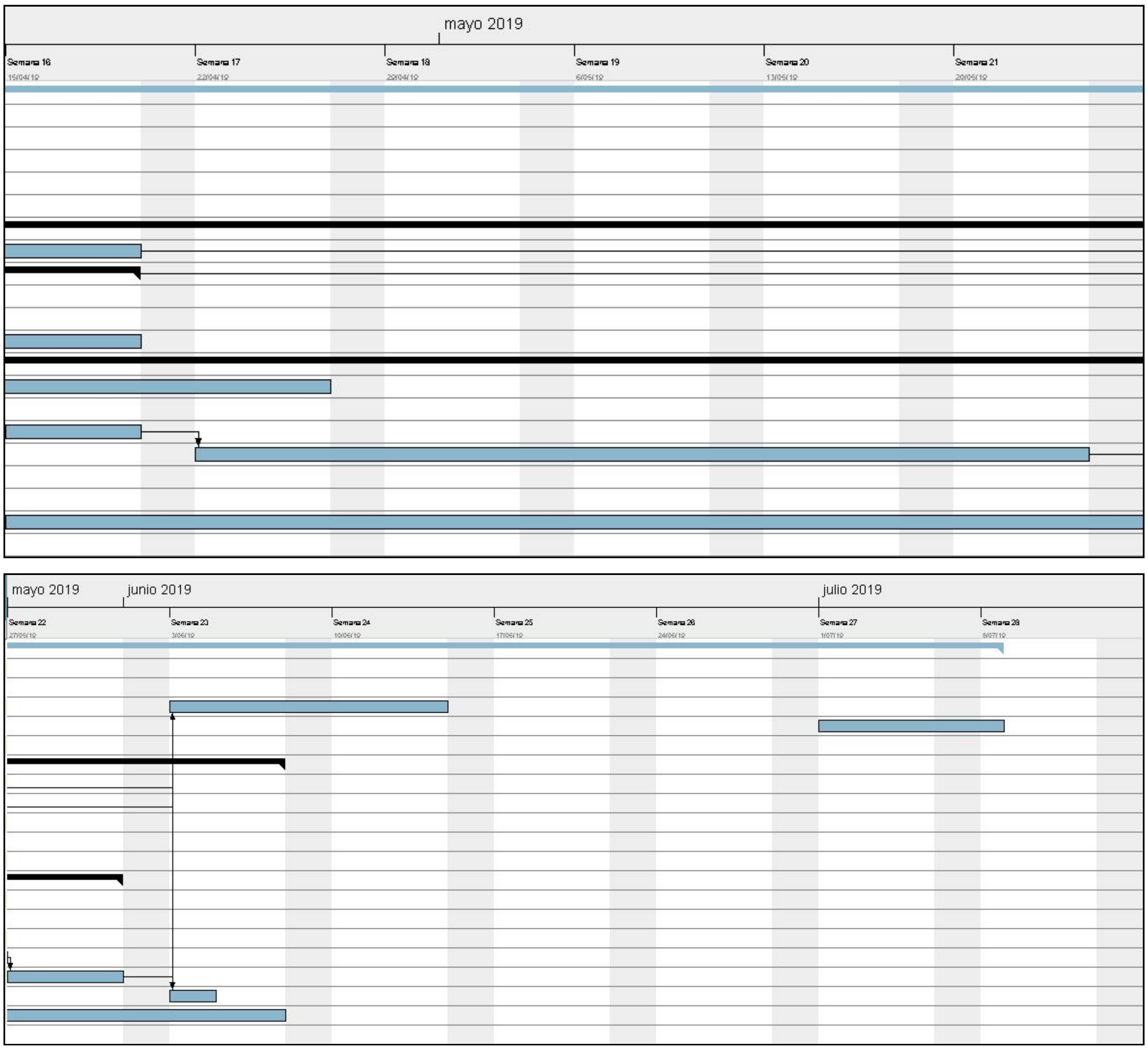


Figure 2.1.1: Gantt diagram of the project

This is translated to a faster speed to code, and less errors or bugs related with the workflow of the game engine.

Although, as can be read between the lines in 2.1 and in Chapter 5, the assessment of planning is poor, a quarter of the project has been left on the road because the programming part was extended more over time than expected. Therefore it has not given time to realize the last part of the features, which was the interaction between NPCs, leaving the project a bit lame.

3

CHAPTER

System analysis and design

In this chapter we will establish the necessary requirements to be able to develop the game at the same time that we explain its structure and relationships.
Just a little clarification to evade confusions, every time we name "NPC" we mean the NPC that can buy, only when we mean "all NPC" we also include the salesman.

3.1 Game Mechanics

As we told in Chapter 1.2 the game tries to simulate a small market, with 3 adjacent shops, where the NPC needs to buy some stuff to hold their two basic

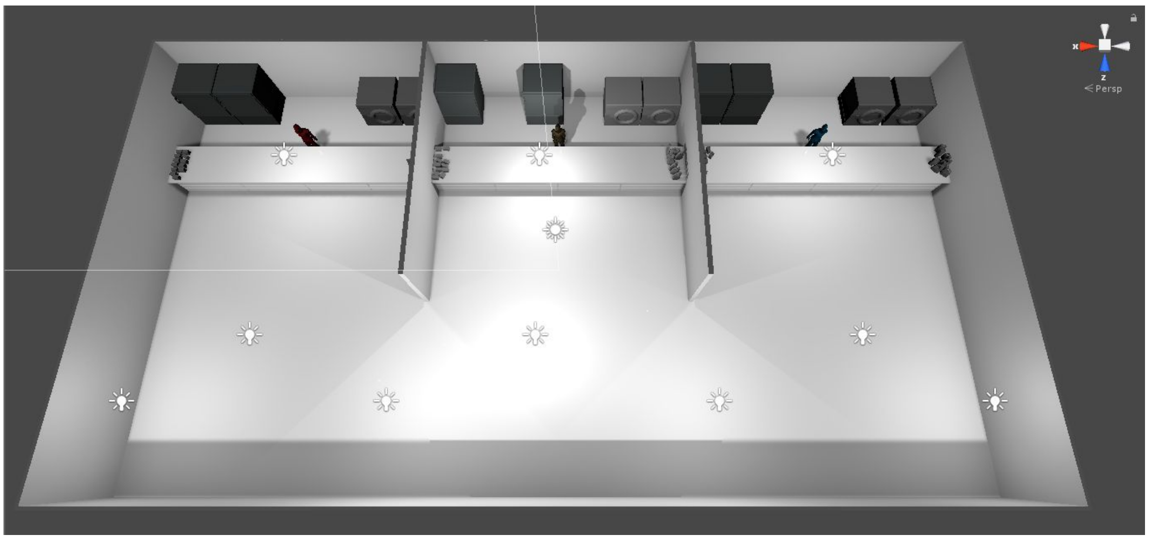


Figure 3.1.1: Shop distribution with the salesman visible

needs satisfied, eating and maintaining their clothes clean. For that, the shops have 4 types of items: food, detergent, washer and fridge. Each one have a specific function and characteristics.

The NPC has to eat food each days. By default the food has a spoil time of one day.

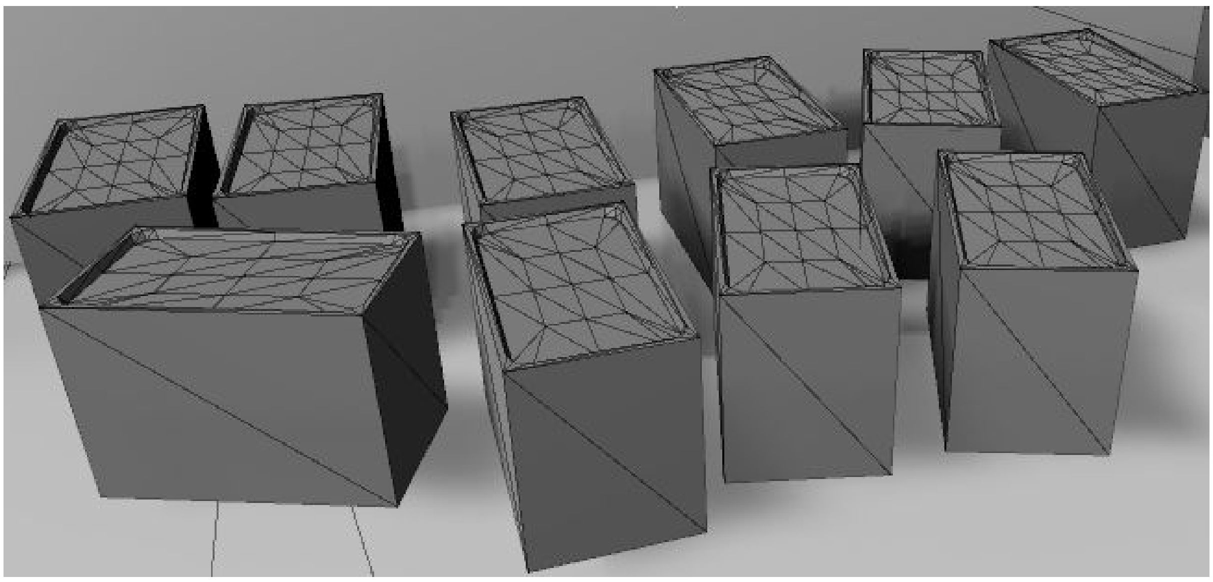


Figure 3.1.2: Food items.

To elong the food spoil time they need to have a fridge, that will increase it around one week. In addition it is necessary to not be hungry on Sunday, because shops are closed that day.

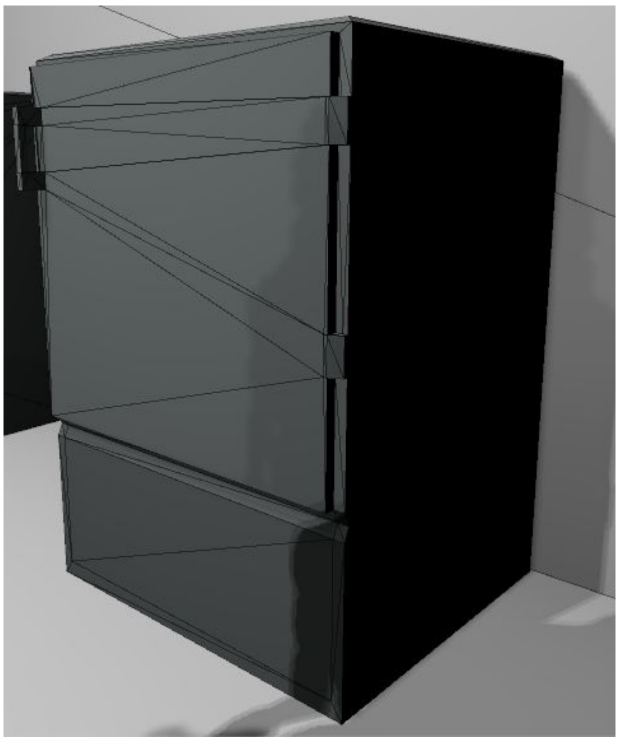


Figure 3.1.3: The fridge.

The NPCs has to clean their clothes, for this they need the washer.

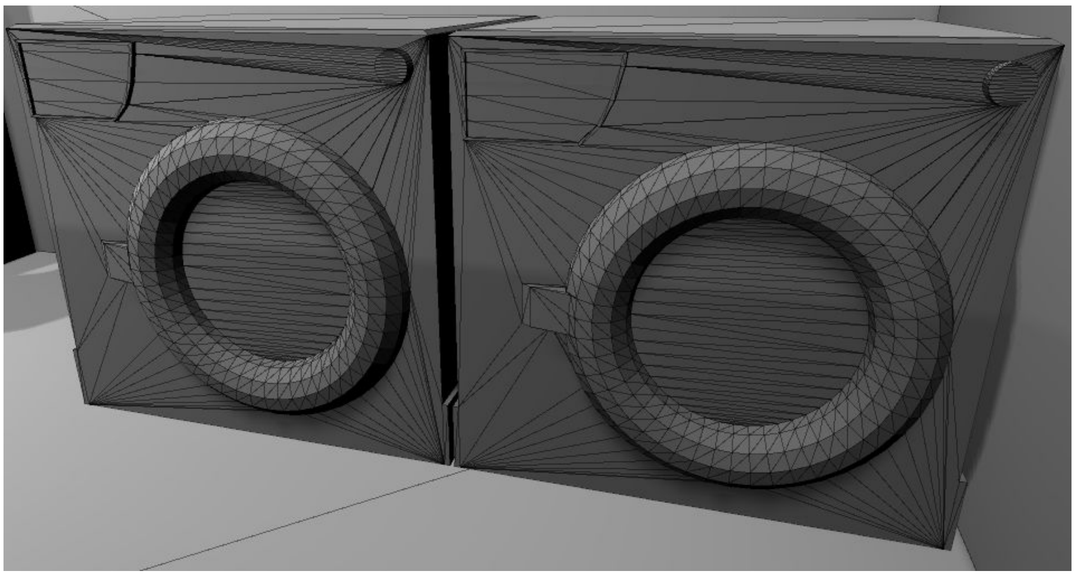


Figure 3.1.4: The washer machine.

Also, he needs detergent to make the washer work. Detergent will determine the result of the clean operation, because if the detergent has bad quality the cloth will be dirty. In addition a bad detergent will lower the washer lifetime.

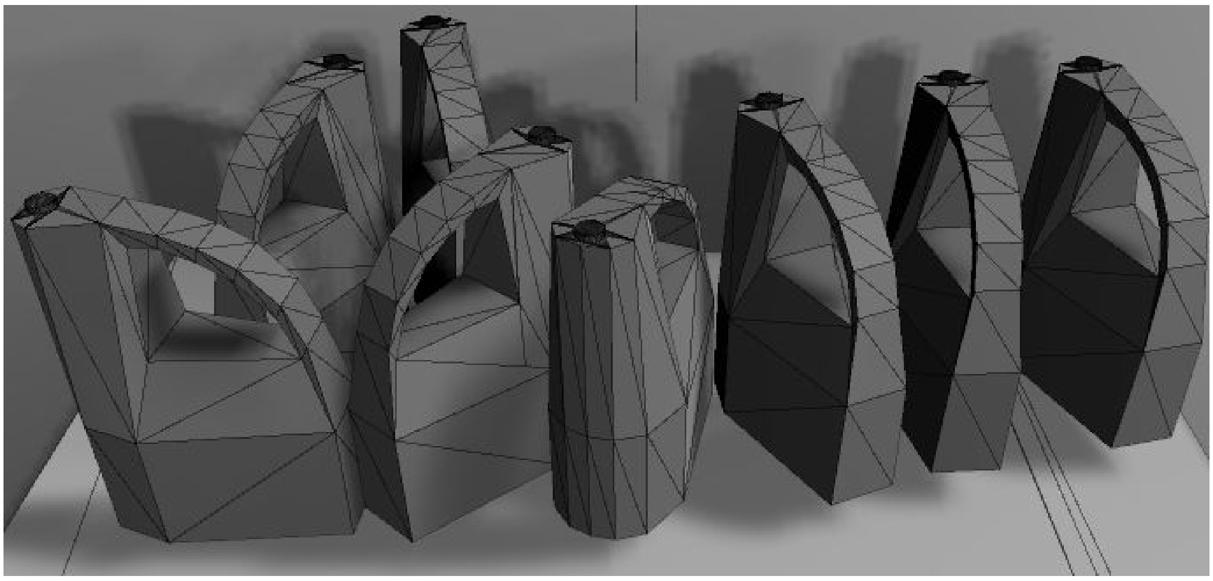


Figure 3.1.5: A bunch of detergents

All the items have three properties in common represented by a number: value, quality and expected lifetime.

- Value indicate the price that the item has.
- Quality is used for quality checks when the items are used.
- Expected lifetime determine when the item is going to break or spoil.

This three properties are determined by the salesman who sells the item (See figure 3.1.6). The three salesman have different quality standards (the maximum and minimum value) with a different benefit margin also. Quality and lifetime are related linearly, depending on the base lifetime of each item and the quality.

Salesman	Personality	Min Quality	Max Quality	Benefit margin	Quality/Price standards
Left	Rude	5	60	10	Cheap
Center	Lonely	55	95	20	Expensive
Right	Sociable	40	85	18	Average

Figure 3.1.6: Salesman properties.

	Food	Detergent	Fridge	Washer
Base Lifetime	7	50	30	20
Base Price	5	10	400	300

Figure 3.1.7: Base price and lifetime of items

Depending on the result of the cleaning of the cloth, the quality of eaten food and the lifetime of the items bought, the relationship between NPCs and salesman will change by a numeric value.

If the NPC could not satisfy his needs due to a washer or fridge break, the relationship with the salesman who has sold this appliance will drop every day while this situation remains unsolved.

NPC will have several characteristics to select the favorite salesman. The first is the personality, all NPCs has a personality and they will try to interact with the salesman they have more affinity with. The result of the interaction will set a multiplier on the relationship weight on the final equation. The second is the relationship with the salesman. The personality define other properties, such as quality/price desired on bought items and the importance of this factor. This is the third factor. The priorities can be shown below in Figure 3.1.9.

The NPC selection equation can be shown in Figure 3.1.10

Personality	BLAB	FRIENDLY	LONELY	PERFECTIONIST	RUDE	SAVER	SELFISH	SOCIABLE
BLAB	1	0.5	0.5		1			
FRIENDLY		1	1		0.5		0.5	1
LONELY	0.5	1		0.5	0.5		0.5	1
PERFECTIONIST						1		
RUDE	1	0.5	0.5				1	0.5
SAVER						1	0.5	1
SELFISH		0.5		1		0.5	1	
SOCIABLE		1			0.5			1

Figure 3.1.8: Interaction modifiers when one NPC (left) interacts with other with another (above) personality.
The empty spaces represents an intermedium value of 0.75.
This values are not represented to remark the special cases.

Moreover to difference the NPC personality at a glance we can see their color code in Figure 3.1.11 and Figure 3.1.12.

Personality	Quality/price preference	Relevance
BLAB	Balance	None
FRIENDLY	Balance	None
LONELY	Balance	None
PERFECTIONIST	Quality	Important
RUDE	Cheap	None
SAVER	Balance	Medium
SELFISH	Cheap	Important
SOCIABLE	Balance	None

Figure 3.1.9: Quality/Price preferences depending on personality.

$$\text{Score} = QP_{Score} + P_{multiplier} * S_{Relationship} + P_{multiplier}$$

Figure 3.1.10: Score equation, when QP_{Score} is the quality price score, being 0 if there is not preference, 4 if the relevance is medium and 10 if the relevance is high. This factor only applies on the salesman with the quality/price desired.
$P_{multiplier}$ is the personality multiplier and $S_{Relationship}$ is the relationship with the salesman.

In the Figure 3.1.10 we can see that NPC with Quality/Price preferences will try to buy the intended items since the begining with their desired Quality/Price, but if the relationship with that salesman decrease, at long term they will buy on another place.

The personality multiplier related directly with the relationship makes more difficult to buy in a shop with a non-affine personality.

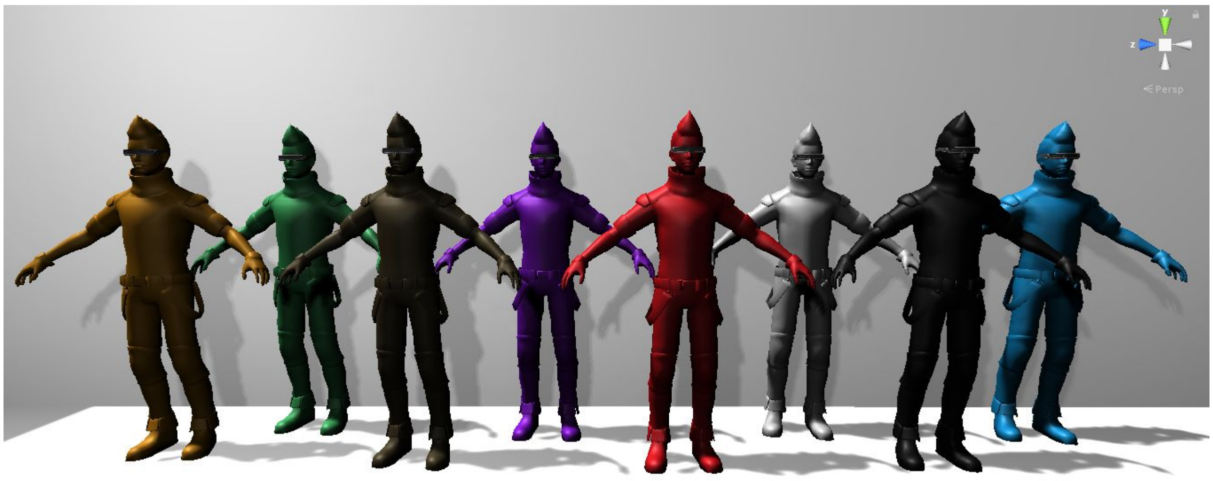


Figure 3.1.11: NPC colors.

Color	Yellow	Green	Brown	Purple	Red	White	Black	Blue
Personality	Blab	Friendly	Lonely	Perfectionist	Rude	Saver	Selfish	Sociable

Figure 3.1.12: NPC relation between color and personality.

Another mechanic in the game is that the NPC can not “see” the items in the shops until they go to check it. For that they will make a queue to talk with the salesman. When they finish this interaction, the NPC can go to check another shop or they can go to buy the items that he needs.

With all this concepts about the game and mechanics, we can show a simple flow of the actions of the NPC in the shop. See the figure 3.1.13 below.

To finish this section, NPC personalities can be useful to introduce unique mechanics to certain types of NPC. As an example of that we have the NPC with perfectionist personality, that always will see all the shops available.

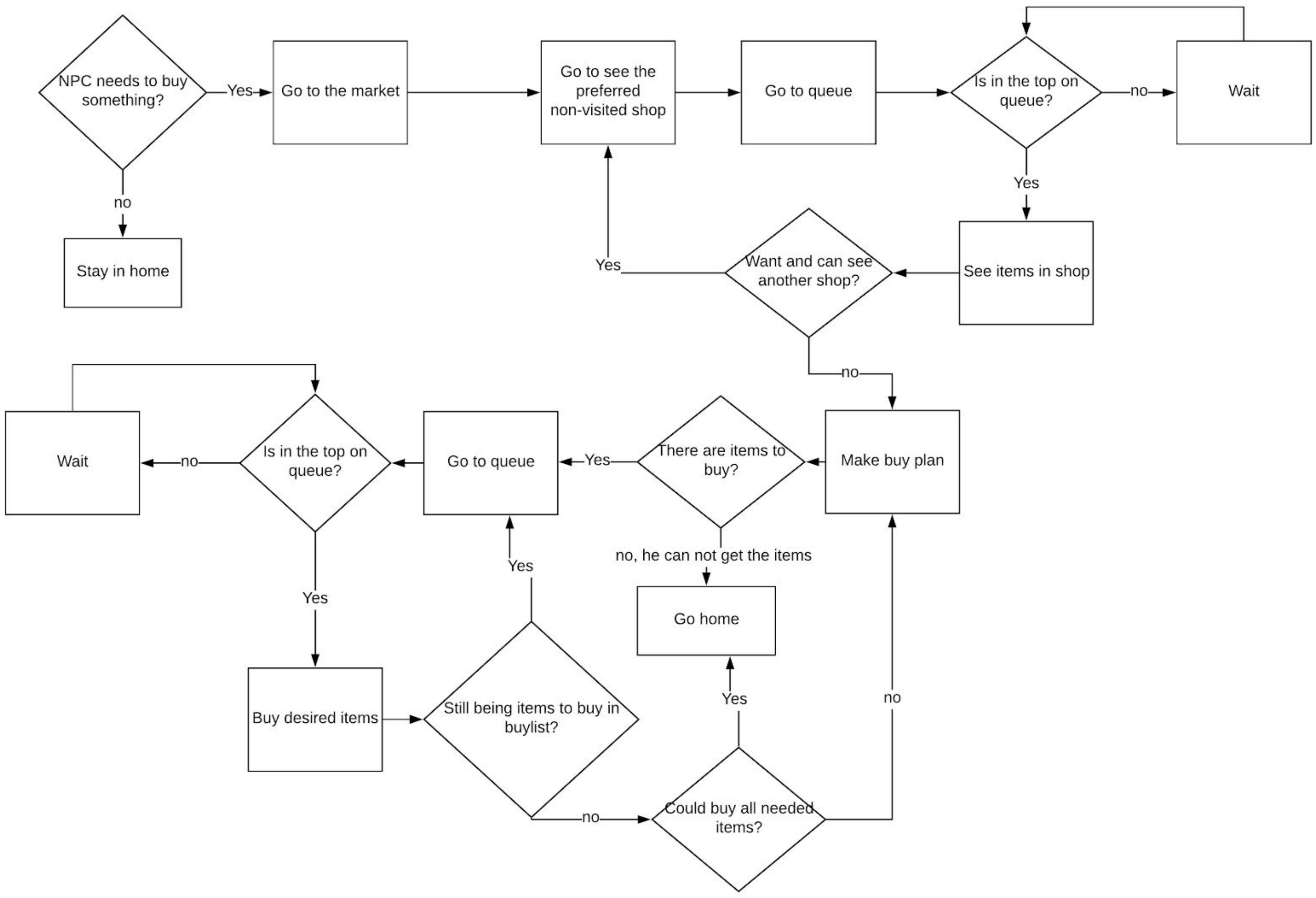


Figure 3.1.13: NPC actions flow.

3.2 Requirements analysis

In this section we will show the functional requirements of the game.

● Name:	**Week cycle**
● Input:	Actual day
● Output;	Next day
● The game simulates week cycle. Shops are closed on Sunday, so the day must skipping to Monday but without skip the events that have happened.	

Figure 3.2.1: Requirement <REQ1 , Week cycle>

- Name: **All NPC are autonomous**
- Input: Environment
- Output; NPC answer
- The salesman and the NPCs must solve all problems related to their functions autonomously, without the intervention of the player.

Figure 3.2.2: Requirement <REQ2 , All NPCs are autonomous>

- Name: **The salesman sells items to npc**
- Input: Salesman with items, NPC without an item(s) that need to satisfy his need. NPC has money to buy them.
- Output; Salesman and NPC trade items for money
- The salesman sells an item or some items to a NPC for money.

Figure 3.2.3: Requirement <REQ3 , Salesman selling>

- Name: **The salesman resupplies the shop**
- Input: End of the day
- Output; Buy needed items for shop. Add them to shop inventory
- At the end of the day the salesman will buy items at the base price and add them to the store inventory with their final price. The salesman will resupply items in the next order: Fridge, Washer, Detergent, Food. In addition, there can not be a difference more than 40% relative to maximum stock between any item. For example, if the shop does not have any washer in stock, salesman can not fill the food stock over 40% of max stock.

Figure 3.2.4: Requirement <REQ4 , Shop Resupply>

- Name: **Salesman bankruptcy**
- Input: Salesman with negative money for three days
- Output; Salesman store closed
- Salesman will close the store and removed from the game

Figure 3.2.5: Requirement <REQ5 , Salesman bankruptcy>

● Name:	**Salesman rent pay**
● Input:	Salesman money, rent value
● Output;	Salesman money
● Salesman will pay every day the rent. His price is fixed to 35.	

Figure 3.2.6: Requirement <REQ6 , Salesman rent>

● Name:	**NPC need, Eat**
● Input:	Food
● Output;	Food(used), salesman relationship change
● The NPC must eat all the days. If he has a fridge can store the food for one week. In other case food will spoil in one day.	

Figure 3.2.7: Requirement <REQ7.1 , NPC need, Eat>

● Name:	**NPC need, Clean cloth**
● Input:	Detergent (used), Washer
● Output;	Clean cloth or dirty cloth, salesman relationship change
● The NPC has three cloths, when all clothes are used he has to clean them. If the detergent has bad quality the cloth will remain dirty and low the relationship. Otherwise cloth will be clean and the relationship will grow a bit.	

Figure 3.2.8: Requirement <REQ7.2 , NPC need, Cloth>

● Name:	**NPC can buy item**
● Input:	NPC need some item, salesman available, money, visited shops
● Output;	NPC buy an item(s) or return home
● The NPC can buy items in any visited shop. This is the other side of REQ3, from the NPC. If the NPC could not get the items, he will return home. The buy flux is described in Figure 3.1.13.	

Figure 3.2.9: Requirement <REQ8 , NPC buy item>

●	Name:	**Items can be broken or expire**
●	Input:	Item
●	Output;	Item destruction
●	When the item is consumed or his lifetime arrives to there max lifetime, the item will be destroyed.	

Figure 3.2.10: Requirement <REQ9 , Item expire>

●	Name:	**Items has different stats**
●	Input:	Item
●	Output;	None
●	All items have quality, max lifetime and different price depending on the salesman who sell it. This salesman will have a range of parameters establishing this values.	

Figure 3.2.11: Requirement <REQ10 , Items stats>

●	Name:	**Items in the shop are visible in scene**
●	Input:	Salesman inventory, Items
●	Output;	Visibility
●	The scene will show the assets of available items in the shop	

Figure 3.2.12: Requirement <REQ11 , Item visibility>

●	Name:	**HUD (Head-Up Interface) available info**
●	Input:	None
●	Output;	None
●	The GUI will show info about the Day and salesman money. In addition, pressing a button (“2”) will show or hide advanced info about NPC items in inventory, relationships and preferred salesman to buy.	

Figure 3.2.13: Requirement <REQ12 , HUD info>

● Name:	**NPC queuing to buy in shops**
● Input:	NPC list in queue of a shop, active npc
● Output;	Change position or start interaction with salesman.
● The NPC will positionate in the last position of the queue buy. The position will be updated when the queue drops. If the NPC is first in queue, he will interact with the salesman.	

Figure 3.2.14: Requirement <REQ13 , shop queues>

● Name:	**NPC can see some shops**
● Input:	NPC visited shops and shops without see, end of see a shop
● Output;	See another shop or make a buy plan
● When the NPC finishes seeing one shop they can go to see another shop. For that, there is a probability of 40%. If the probability determines that do not see another shop or he has already see all available shops, the NPC will make the buy plan.	

Figure 3.2.15: Requirement <REQ14 , NPC see shops>

● Name:	**All NPC has different personalities**
● Input:	None
● Output;	None
● All NPC will have different personalities that will affect the selection preferences of NPC to choose a salesman more accorded to their personality, and other special characteristics.	

Figure 3.2.16: Requirement <REQ15 , All NPC have personalities>

● Name:	**Personality special characteristic. Perfectionist**
● Input:	None
● Output;	None
● The perfectionist NPC will see all the shops available before making the buy plan.	

Figure 3.2.17: Requirement <REQ15.1 ,Perfectionist special abilities>

3.3 AI Architecture

In this section we are going to talk about what challenges AI has to solve in this game.

We can differentiate clearly two different parts, on the one hand the salesman and on the other hand the NPC.

The behaviour of the salesman AI is relatively simple, he just must take care of two functions. Maintaining the store supplied with objects and carrying out sales operations. The salesman do not have incorporated AI techniques that aim to obtain the maximum income.

However, the NPC are the most important part of the AI of the game, as they are able to perform several actions and must be able to manage all possible problems.

The functions to be performed using the AI in NPC are:

- Needs assessments.
- Evaluate when to go to the store.
- Check the prices and stock of the store.
- Buy in the store.
- Make a buy strategy.
- NPC movement.
- Modify the relationship with salesman according to events.

The challenges that AI must overcome due to its functionalities are:

- Manage the situation when the queue is full.
- Select the most suitable salesman.
- Select other salesman if:
    - There is no stock.
    - Not enough money available.
    - Shop has been closed.
- The selection of other dependents when deciding the purchase strategy should only occur if you have visited the store.
- Return home in case you cannot meet their needs.
- Movement management, and its relationship to the environment and other variables.

4

CHAPTER

Work development and results

In this chapter we will talk about how the project has been done in detail and the results that have been achieved.

4.1 Work development

The development of the game revolves around 3 classes, WorldState, NPC and Salesman. Each of these classes has a variety of satellite classes that provide functionalities and behaviors.

The general class diagram can be seen at the end of this section Figure 4.1.17 [8]. But to make more understandable the Classes and the functions explanations, we will use partial Class diagrams to make it more clear.

WorldState Class

Let's start with *WorldState*, a class formed by a Singleton and which is the organizational axis of the game.
In this class we have lists with the *prefabs** of the NPCs and Salesman, which can be used at the beginning of the game to spawn them.

Also, we have the public lists with the NPCs and the salesmans in the scene. This lists are used to perform searches and operations with other methods or functions.

[*] A prefab is a gameobject saved inside Unity with some components attached, as Transform (position, rotation and scale), Mesh, Materials, Colliders, Scripts, etc. The big advantage is that we can spawn it directly in scene, because we can left them ready to use.

An important detail of C# and Unity, is that the NPCs saved in the list always are accessible, even if they are disabled in the scene. This facilitates a lot of work in implementation because we can use the disable NPC and make any operation that we want if we access from the list. For example, we can call the functions related with the use of food for all the NPC at the end of the day, just moving in with a loop and calling the necessary functions.

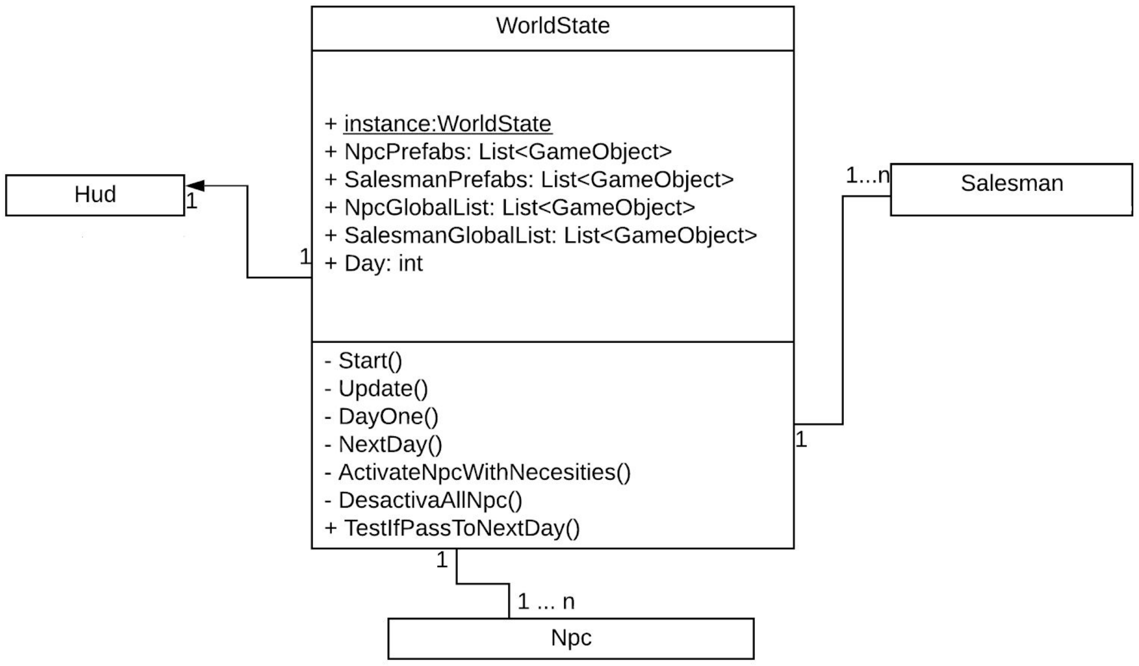

WorldState
+ instance:WorldState
+ NpcPrefabs: List<GameObject>
+ SalesmanPrefabs: List<GameObject>
+ NpcGlobalList: List<GameObject>
+ SalesmanGlobalList: List<GameObject>
+ Day: int
- Start()
- Update()
- DayOne()
- NextDay()
- ActivateNpcWithNecesities()
- DesactivaAllNpc()
+ TestIfPassToNextDay()
Hud
1
1
Salesman
1...n
1
1
1 ... n
Npc

Figure 4.1.1: WorldState Class

In multiple classes we have two special methods, *NextDay*() and *DayOne*(), the first one takes care of all the changes that need to be made an to be check when changing the day, and the second one takes care of initializing the game on the first day.

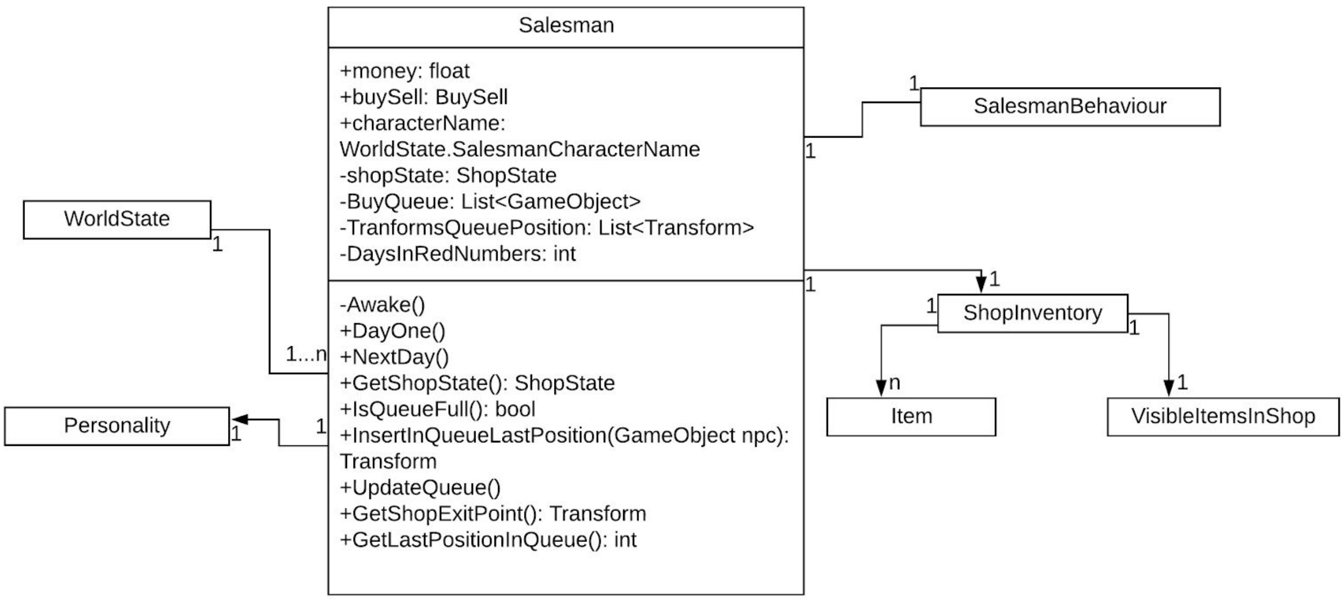
This is especially important in WorldState class because *DayOne*() orders the initialization of all the elements of the game.
NextDay() in this class makes the change of day, making all the necessary calls to all elements of the game. Lastly, this class activates those NPC that need to buy and go to the market on the next day. *NextDay*() is called automatically when the last NPC go out of the store and in Sunday.

Also, this class has inputs for the only actions that can be performed by the user, which are to force the passage of the day (pressing “1”) and turn on or off the HUD (pressing “2”).

Salesman Class

The Salesman has five satellite classes in its gameObject.

The main class, Salesman, keep the waiting positions off the purchase of queue, a reference to the NPCs in the salesman queue, his money and the bunch of properties that determine the type of Salesman.



Salesman
+money: float
+buySell: BuySell
+characterName: WorldState.SalesmanCharacterName
-shopState: ShopState
-BuyQueue: List<GameObject>
-TranformsQueuePosition: List<Transform>
-DaysInRedNumbers: int
-Awake()
+DayOne()
+NextDay()
+GetShopState(): ShopState
+IsQueueFull(): bool
+InsertInQueueLastPosition(GameObject npc): Transform
+UpdateQueue()
+GetShopExitPoint(): Transform
+GetLastPositionInQueue(): int
SalesmanBehaviour
WorldState
Personality
ShopInventory
Item
VisibleItemsInShop
1
1...n
n

Figure 4.1.2: Salesman Class

We have several methods that are in charge of positioning the salesman correctly in the store, along with functions for the correct functioning of the queue, like *isQueueFull()*, *UpdateQueue()*, *InsertInQueueLastPosition(Gameobject npc)*. The name of this functions are self-describing.

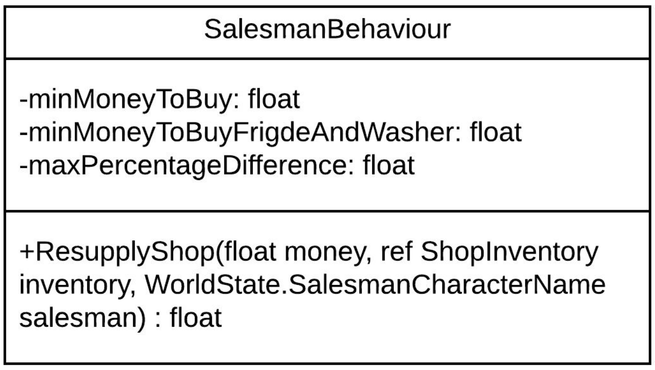
In Salesman class, *DayOne()* send orders to the satellite classes to supply the store on the first day.

NextDay() takes care of subtracting the shop rent every day, checking if it has gone bankrupt and resupply the store if there is enough money.

Here ends the explanation of the main class, now we are going to talk about the satellite classes.

First we have SalesmanBehaviour, which contains the constants of the maximum quantity of objects that can be keeped in the inventory and is in charge of carrying out the resupply of the store.

Resupply the store has an order of priority, as we say in Chapter 3.1. First, the salesman attempts to resupply fridges and washers, and if it is not possible, the detergent and food will be resupplied. There are two minimum amounts of money that the salesman needs to resupply the store, one for electrodomestics and other for global. This is designed to prevent the salesman from getting ruined if he has little money and nobody buys from him in the next few days. This quantities are fixed to 800 in the first case and 150 in the second.

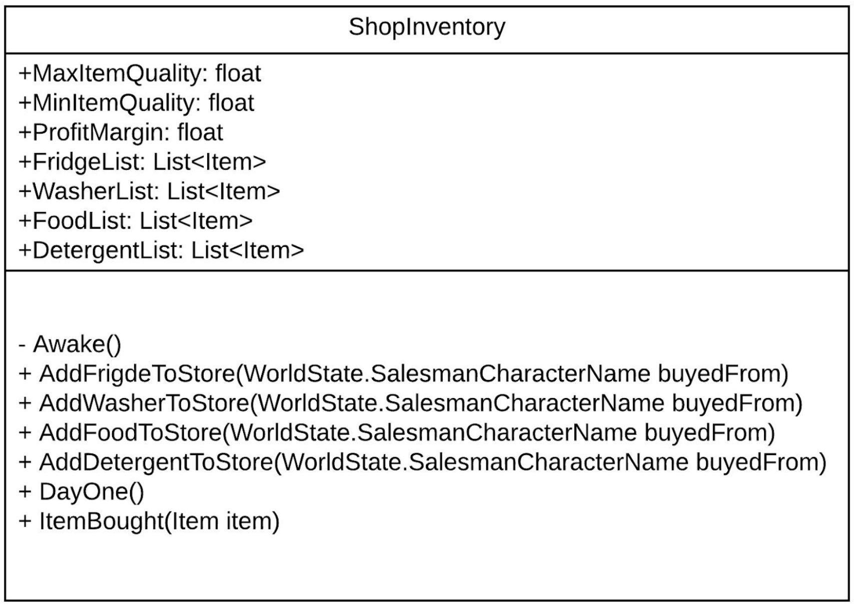


```
SalesmanBehaviour

-minMoneyToBuy: float
-minMoneyToBuyFrigdeAndWasher: float
-maxPercentageDifference: float

+ResupplyShop(float money, ref ShopInventory
inventory, WorldState.SalesmanCharacterName
salesman) : float
```

Figure 4.1.3: SalesmanBehaviour Class

In addition, salesmen can not buy all items of one type at once to secure that dispose of all types of items. For example, if the dependant has 40% of the maximum food stock and 0% of refrigerators, he will not be able to buy more food until he buys more refrigerators. This difference limit is set at 40% and is customizable.
This class leaves freedom to include new AI behaviors on the salesman.

As a second satellite class we have ShopInventory, which is responsible of storing all the items in the store. It also determines some qualities of the salesman, such as the maximum and minimal quality of the objects and the profit margin. As well, this class has several necessary constants, such as the basic prices of items and the rent price.

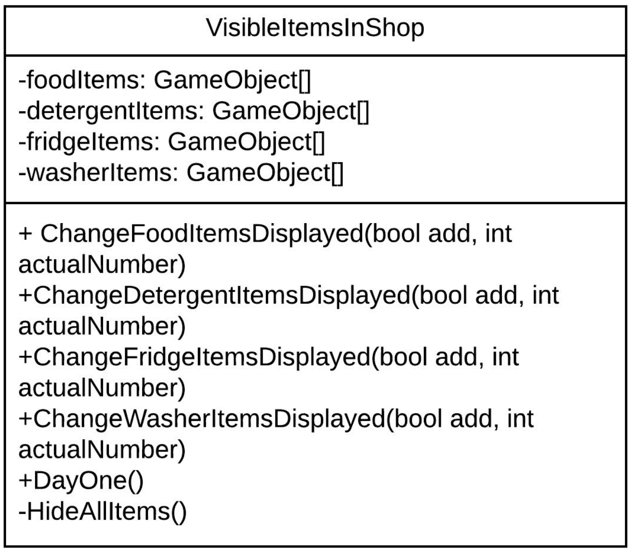


```
ShopInventory

+MaxItemQuality: float
+MinItemQuality: float
+ProfitMargin: float
+FridgeList: List<Item>
+WasherList: List<Item>
+FoodList: List<Item>
+DetergentList: List<Item>

- Awake()
+ AddFrigdeToStore(WorldState.SalesmanCharacterName buyedFrom)
+ AddWasherToStore(WorldState.SalesmanCharacterName buyedFrom)
+ AddFoodToStore(WorldState.SalesmanCharacterName buyedFrom)
+ AddDetergentToStore(WorldState.SalesmanCharacterName buyedFrom)
+ DayOne()
+ ItemBought(Item item)
```

Figure 4.1.4: ShopInventory Class

ShopInventory has the functions related with adding new objects to the store, where using the properties of Salesman, the final objects are created with a random quality within the established values, and a durability according to the quality. Using the profit margin of salesman and the base price of the item, the final price is setted.

ShopInventory also has an associated class, VisibleItemsInShop, which displays inventory items on the scene.



VisibleItemsInShop

-foodItems: GameObject[]
-detergentItems: GameObject[]
-fridgeItems: GameObject[]
-washerItems: GameObject[]

+ ChangeFoodItemsDisplayed(bool add, int actualNumber)
+ChangeDetergentItemsDisplayed(bool add, int actualNumber)
+ChangeFridgeItemsDisplayed(bool add, int actualNumber)
+ChangeWasherItemsDisplayed(bool add, int actualNumber)
+DayOne()
-HideAllItems()

Figure 4.1.5: VisibleItemsInShopClass

The VisibleItemsInShop class stores the meshes of the items positioned in the shop, and only shows the same number of elements of one determined type available in ShopInventory. Due to that, this class is not related directly with the item class, because there is a disconnection between Item class and the items showed.

To finish the Salesman, he also has the Personality class, which gives the personality to the Salesman and the base movement script of Unity, although it is not used for anything because the salesman remains static in his shop configured position.

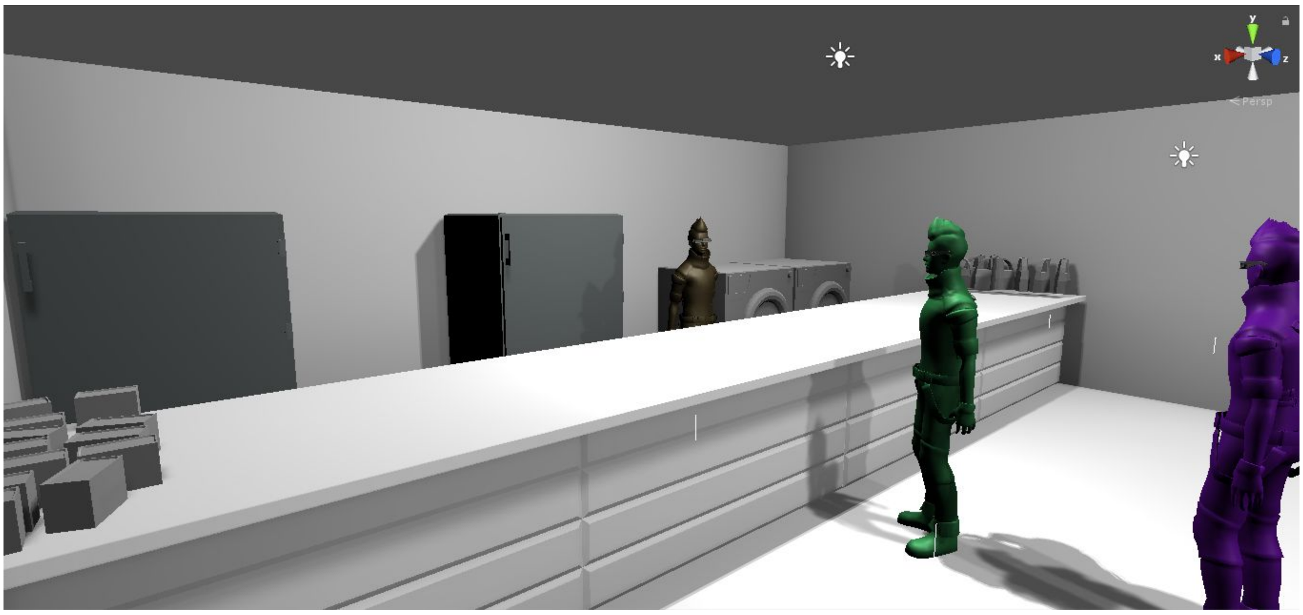


Figure 4.1.6: Salesman screenshot when selling items

NPC Class

This is the most complex entity of all. We recommend to check again Figure 3.1.13 because with the flow diagram in the hand it is more easy to understand this class in the first try.

In the main class we have several functions that coordinate the work of the others. The first important function is *GoToShop()* which analyzes whether the NPC should go to the market to buy. This is done by analysing the items in the inventory against the minimum quantity that should be available. The money is checked because maybe the NPC may not have enough money to buy the items. If the result is true, the NPC will do appear (the gameobject will be activated by WorldState) in the store the next day.

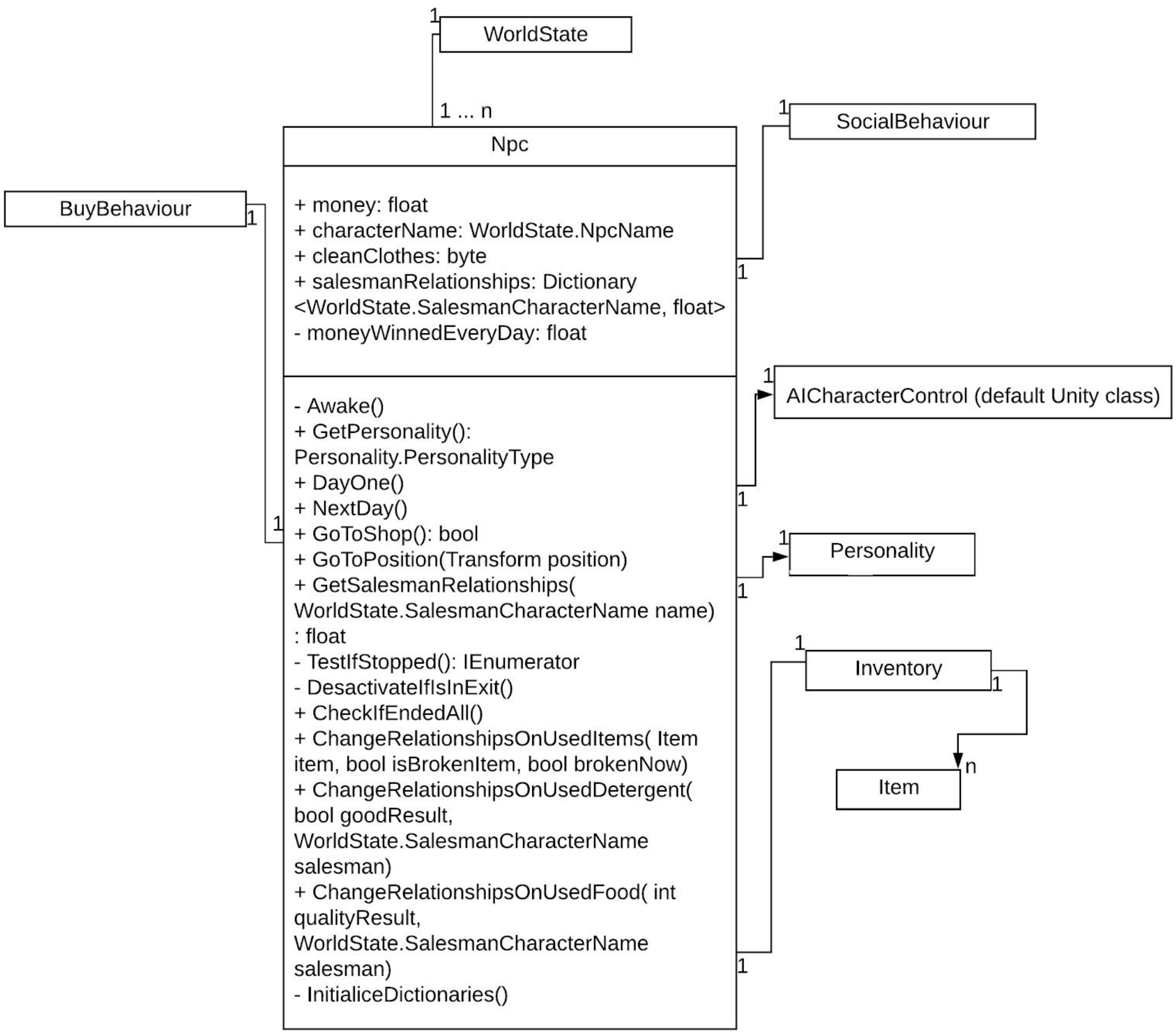

WorldState
1
1 ... n
Npc
+ money: float
+ characterName: WorldState.NpcName
+ cleanClothes: byte
+ salesmanRelationships: Dictionary <WorldState.SalesmanCharacterName, float>
- moneyWinnedEveryDay: float
- Awake()
+ GetPersonality(): Personality.PersonalityType
+ DayOne()
+ NextDay()
+ GoToShop(): bool
+ GoToPosition(Transform position)
+ GetSalesmanRelationships(WorldState.SalesmanCharacterName name) : float
- TestIfStopped(): IEnumerator
- DesactivateIfIsInExit()
+ CheckIfEndedAll()
+ ChangeRelationshipsOnUsedItems(Item item, bool isBrokenItem, bool brokenNow)
+ ChangeRelationshipsOnUsedDetergent(bool goodResult, WorldState.SalesmanCharacterName salesman)
+ ChangeRelationshipsOnUsedFood(int qualityResult, WorldState.SalesmanCharacterName salesman)
- InitialiceDictionaries()
BuyBehaviour
SocialBehaviour
AICharacterControl (default Unity class)
Personality
Inventory
Item
1
n

Figure 4.1.7: NPC class

Inside NPC class is another group of functions that manages the movement orders, the most relevant is *TestIfStopped()*. It is a coroutine that ends when the NPC reaches its destination by developing a series of responses according to the available information and the status of the satellite classes.

Also, we have the set of functions that take care of changing the relationship with the salesman. There are three types according to what has happened. The first, *ChangeRelationshipsOnUsedFood()* change the relationship between the NPC and the salesman according to the quality of the food consumed, simply modifying the relationship by adding a different constant according to the quality.

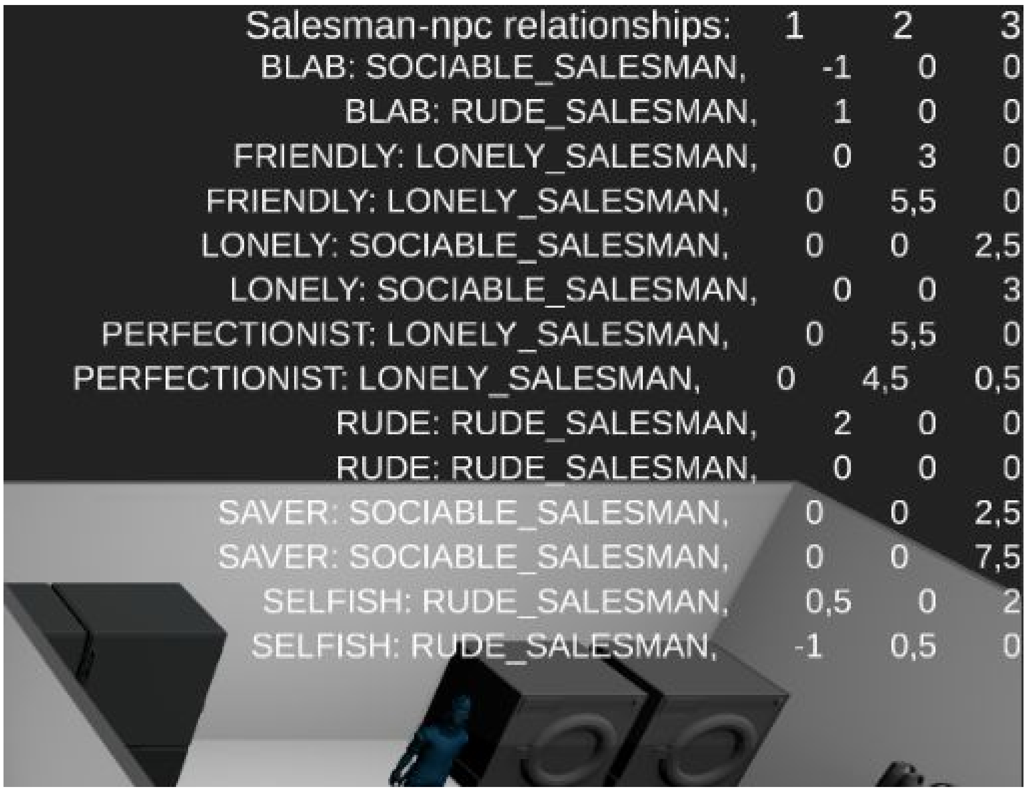
	Positive answer	Neutral answer	Bad answer
Relationship change	2	0.5	-1
Quality value of the item	Quality >= 65	65 > Quality > 35	Quality <= 35

Figure 4.1.8: Relationship answer related with quality

The second, *ChangeRelationshipsOnUsedDetergent()*, is responsible for changing the relationship after washing clothes. If the cloth gets dirty because the detergent was bad, the relationship goes down and if it goes clean it will grown with the neutral value.

To calculate this result we use a random value between 0 and 100 and we tackle it with the double of the quality of the detergent. If the random value is bigger, the clothes are dirty. Note that if the clothes are dirty, the next day the NPC will have to try to clean the cloths again.

It should be remembered that a bad detergent will also reduce the estimated life of the washing machine.



```
Salesman-npc relationships:   1     2     3
      BLAB: SOCIABLE_SALESMAN,    -1     0     0
          BLAB: RUDE_SALESMAN,     1     0     0
     FRIENDLY: LONELY_SALESMAN,    0     3     0
     FRIENDLY: LONELY_SALESMAN,    0   5,5     0
     LONELY: SOCIABLE_SALESMAN,    0     0   2,5
     LONELY: SOCIABLE_SALESMAN,    0     0     3
PERFECTIONIST: LONELY_SALESMAN,    0   5,5     0
PERFECTIONIST: LONELY_SALESMAN,    0   4,5   0,5
          RUDE: RUDE_SALESMAN,     2     0     0
          RUDE: RUDE_SALESMAN,     0     0     0
      SAVER: SOCIABLE_SALESMAN,    0     0   2,5
      SAVER: SOCIABLE_SALESMAN,    0     0   7,5
        SELFISH: RUDE_SALESMAN,  0,5     0     2
        SELFISH: RUDE_SALESMAN,   -1   0,5     0
```

Figure 4.1.9: Relationships values and prefered salesman can be shown if we press the key "2"

The last one is *ChangeRelationshipsOnUsedItems()*, which takes care of when an object is broken or spoiled. Moreover this function as well takes into account if the NPC has not been able to meet any of his needs because the fridge or the washing machine was broken. In this case, the negative relationship modificator will be applied each day until the NPC purchase that item again. This negative is more probable to be applied on Sundays, because the shops are closed.

The rest of the function takes care of the cases where the objects are broken. If the object has been of good quality, the positive multiplier will be added to the relationship multiplied by the number of extra days the object has lasted. It happens the same if the object is of poor quality, just applying the negative modifier.

DayOne() is in charge of initializing the relationships dictionaries and giving the order to initialize the inventory.

NextDay() is in charge of carrying out the needs at the end of the day of the NPC and check all the objects if they have been broken. In addition, this function add a fixed value of money to the NPC every day.

As satellite classes, we have Personality, Inventory, SocialBehaviour and BuyBehaviour.

In Inventory class we have stored the NPC items and his money. We use *AddNewItem()* to insert the new items bought in the attributes. We also have *UpdateInventory()* that takes care of at the end of the day checks if the objects have been broken or expired. Also we have *UseFood()* and *CleanCloth()* to run checks to test the behavior of the food and cloth that we talked about before.

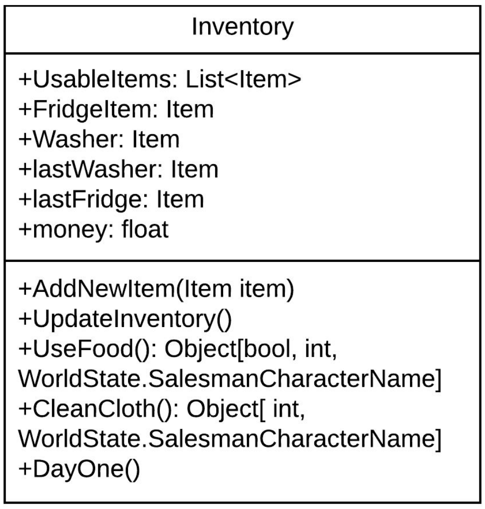


Figure 4.1.10: Inventory class

We have in this class *DayOne()*, that is in charge of adding half used items to the player at the beginning of the game. So the NPCs will have to buy little by little new items, evading a sudden start in the game. These objects provided by *DayOne()* the boughtFrom field is assigned to a STARTING_ITEM salesman (an inexistent salesman), so when this items breaks they will not affect the relationship with any salesman.

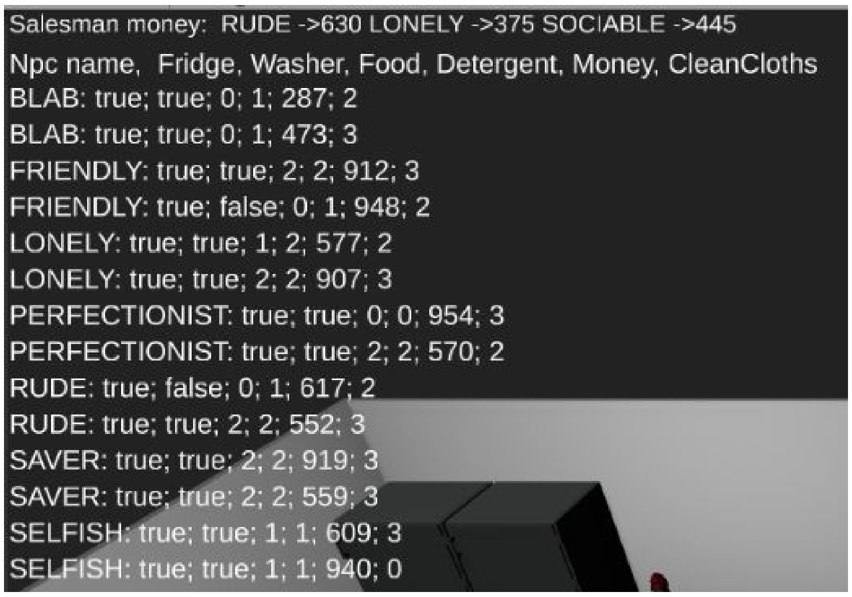

```
Salesman money:  RUDE ->630 LONELY ->375 SOCIABLE ->445
Npc name,  Fridge, Washer, Food, Detergent, Money, CleanCloths
BLAB: true; true; 0; 1; 287; 2
BLAB: true; true; 0; 1; 473; 3
FRIENDLY: true; true; 2; 2; 912; 3
FRIENDLY: true; false; 0; 1; 948; 2
LONELY: true; true; 1; 2; 577; 2
LONELY: true; true; 2; 2; 907; 3
PERFECTIONIST: true; true; 0; 0; 954; 3
PERFECTIONIST: true; true; 2; 2; 570; 2
RUDE: true; false; 0; 1; 617; 2
RUDE: true; true; 2; 2; 552; 3
SAVER: true; true; 2; 2; 919; 3
SAVER: true; true; 2; 2; 559; 3
SELFISH: true; true; 1; 1; 609; 3
SELFISH: true; true; 1; 1; 940; 0
```

Figure 4.1.11: NPC number of items in inventory can be shown if we have the HUD enabled.

In SocialBehaviour we have the calculation of the interaction modifiers according to the personality of the NPCs. This calculation is used to determine what is the Salesman chosen to purchase according to personality. As well, there is the skeleton of a series of unincorporated mechanics (explained in Chapter 5.1).

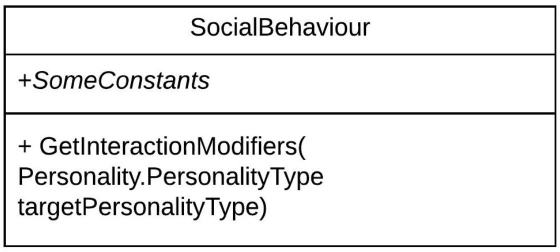


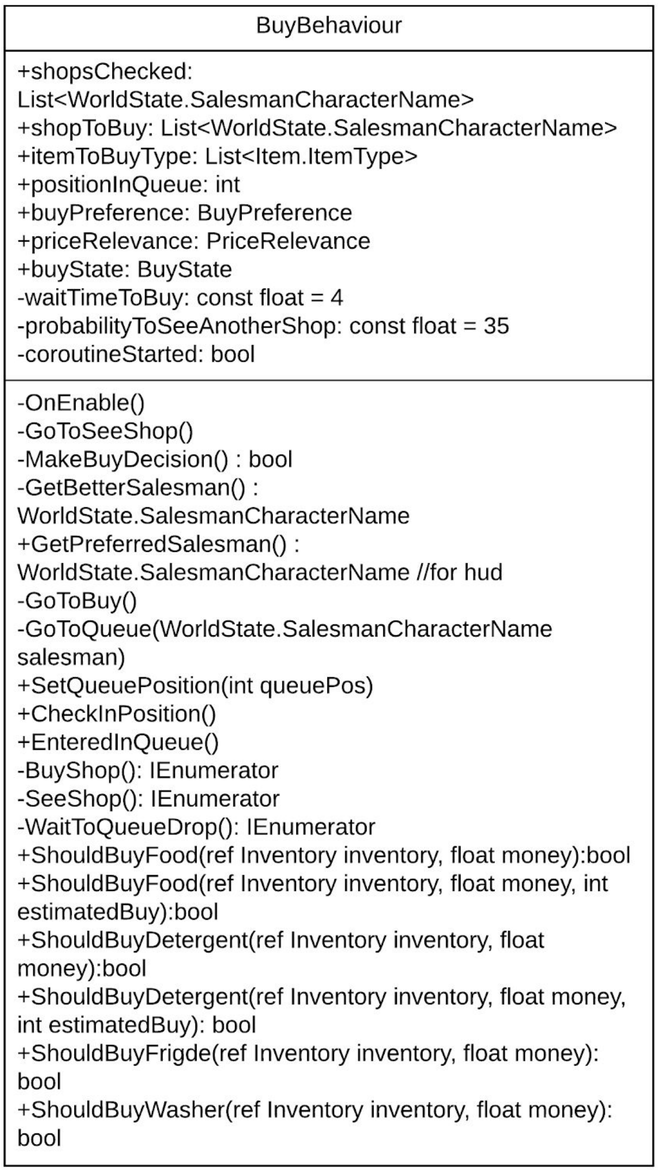
Figure 4.1.12: SocialBehaviour class

Finally we have BuyBehaviour. This class takes care of all the buying behavior of the NPC. Here we have the purchase preferences of the NPC, being able to select if the NPC prefers to buy cheap, quality, or an intermedium value, besides the importance that is given to this.

When NPC needs to buy something, the first thing it does is visiting the store of that salesman who has preference with. This is calculated in the *GetBetterSalesman*() function, where each Salesman is evaluated by the equation shown in Figure 3.1.10, selecting the salesman with the highest score.

With the three factors in the equation choose the Salesman. The first factor is much more important at the beginning, so if an NPC has a preference for a price range whenever he can, he will go for that range. The second factor determines the long term relationship, and the satisfaction with the dependent according to the quality of the received objects. The third factor is a small addiction to eliminate the initial case where the NPC has no preference for relationship or price.

After this choice, the NPC will go to the queue of the store using *GoToQueue()* and as he advance in the queue, the function *CheckInPosition()* will be activated until he gets the first place. When this happens, the coroutine *SeeShop()* will start, adding the store to the list of visited shops. Then the NPC will run a chance to visit another store, in the case that the NPC personality is perfectionist will see all the shops. If the result of the chance is positive, the NPC will visit a different shop, otherwise the NPC will execute the purchase plan, calling *MakeBuyDecision()*.



```
BuyBehaviour
---
+shopsChecked:
List<WorldState.SalesmanCharacterName>
+shopToBuy: List<WorldState.SalesmanCharacterName>
+itemToBuyType: List<Item.ItemType>
+positionInQueue: int
+buyPreference: BuyPreference
+priceRelevance: PriceRelevance
+buyState: BuyState
-waitTimeToBuy: const float = 4
-probabilityToSeeAnotherShop: const float = 35
-coroutineStarted: bool
---
-OnEnable()
-GoToSeeShop()
-MakeBuyDecision() : bool
-GetBetterSalesman() :
WorldState.SalesmanCharacterName
+GetPreferredSalesman() :
WorldState.SalesmanCharacterName //for hud
-GoToBuy()
-GoToQueue(WorldState.SalesmanCharacterName
salesman)
+SetQueuePosition(int queuePos)
+CheckInPosition()
+EnteredInQueue()
-BuyShop(): IEnumerator
-SeeShop(): IEnumerator
-WaitToQueueDrop(): IEnumerator
+ShouldBuyFood(ref Inventory inventory, float money):bool
+ShouldBuyFood(ref Inventory inventory, float money, int
estimatedBuy):bool
+ShouldBuyDetergent(ref Inventory inventory, float
money):bool
+ShouldBuyDetergent(ref Inventory inventory, float money,
int estimatedBuy): bool
+ShouldBuyFrigde(ref Inventory inventory, float money):
bool
+ShouldBuyWasher(ref Inventory inventory, float money):
bool
```

Figure 4.1.13: BuyBehaviour class

The execution of the purchase plan tries to buy all possible objects with the available money, following the preference order of salesmans. The order of priority of the purchase is fridge, washing machine, food and detergent.

The buy list is saved in two queues, one with the item type to buy and other with the selected salesman.

After making the selection, the NPC uses the *GoToBuy()* function, which takes NPC to the queue of the first store that he has chosen. Once he get to the first position of the shop queue the coroutine *BuyShop()* will start, which causes the NPC to buy all the items that the NPC want to buy in that store dequeuing them from both lists.

When he finished this buy, the NPC will go to the other stores where he have to buy something.

The NPC will finished when the lists are empty. Then we have to check if the NPC has all the items that he needs. We have to do that because during the purchase can happen an error. It is possible that one of the objects that this NPC wants to buy has been bought by another npc and the store runs out of stock.

To solve this problem, the purchase plan will be executed again. If a solution is found, the NPC will go to the store to finish buying everything, otherwise he will go home.

HUD Class

The HUD class displays on screen information about the money of the salesman, their relations with the NPC, the objects that the NPCs have and the current day.

It is possible to disable the information that occupies most of the screen by pressing the "2" key, leaving only the current day and money of the salesmans visible.

The HUD is updated when any changes happens in the game. To update it we just we need to seed an update order to the WorldState and he will send it to the HUD class.

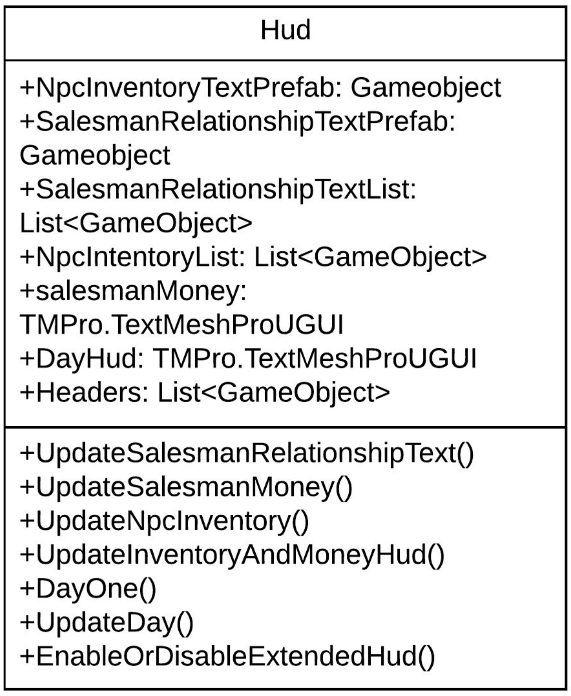

Hud
+NpcInventoryTextPrefab: Gameobject
+SalesmanRelationshipTextPrefab: Gameobject
+SalesmanRelationshipTextList: List<GameObject>
+NpcIntentoryList: List<GameObject>
+salesmanMoney: TMPro.TextMeshProUGUI
+DayHud: TMPro.TextMeshProUGUI
+Headers: List<GameObject>
+UpdateSalesmanRelationshipText()
+UpdateSalesmanMoney()
+UpdateNpcInventory()
+UpdateInventoryAndMoneyHud()
+DayOne()
+UpdateDay()
+EnableOrDisableExtendedHud()

Figure 4.1.14: HUD class

Personality Class

The personality class only has a definition for the personality type. No other functions are here right now, but can be used in the future to define unique behaviors based on the personality.

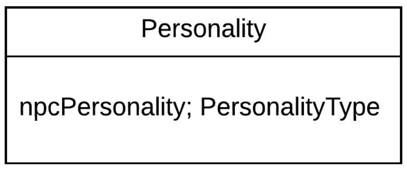


Figure 4.1.15: Personality class

Item Class

The item class represents the properties of the elements bought in the store. Has an itemType property, value, quality, expected lifetime and the salesman who sell it.

This class has two constructors. The first creates a new item with determined stats. The second constructor creates an item with random stats. The first constructor is used when a salesman buys a new item to add it to store, and the other is used to create the initial items of the NPCs.

The *NextDay()* method increase by one the actual lifetime of all the items.

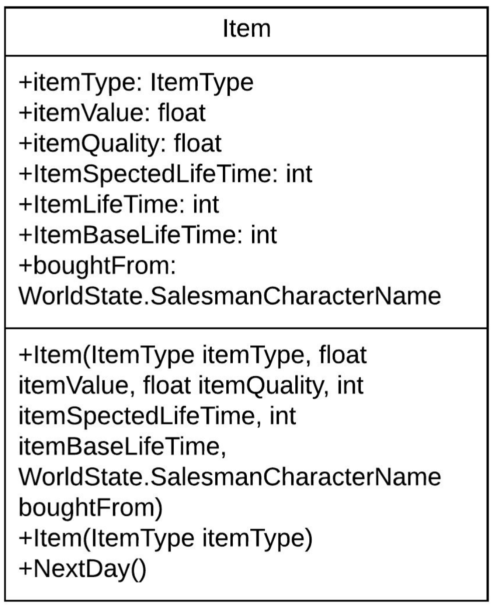

Item
+itemType: ItemType
+itemValue: float
+itemQuality: float
+ItemSpectedLifeTime: int
+ItemLifeTime: int
+ItemBaseLifeTime: int
+boughtFrom: WorldState.SalesmanCharacterName
+Item(ItemType itemType, float itemValue, float itemQuality, int itemSpectedLifeTime, int itemBaseLifeTime, WorldState.SalesmanCharacterName boughtFrom)
+Item(ItemType itemType)
+NextDay()

Figure 4.1.16: Item class

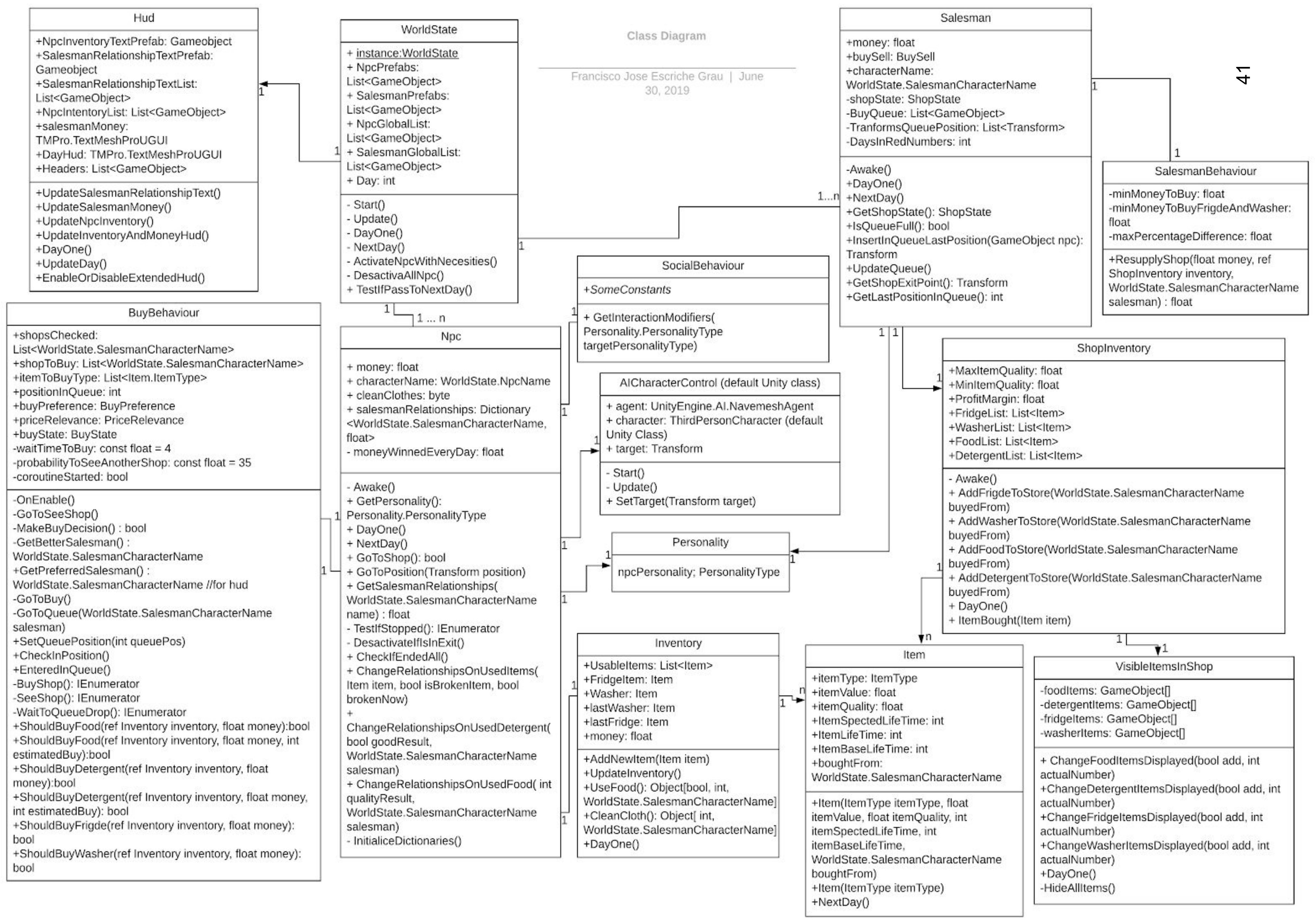

Class Diagram
Francisco Jose Escriche Grau | June 30, 2019
Hud
+NpcInventoryTextPrefab: Gameobject
+SalesmanRelationshipTextPrefab: Gameobject
+SalesmanRelationshipTextList: List<GameObject>
+NpcIntentoryList: List<GameObject>
+salesmanMoney: TMPro.TextMeshProUGUI
+DayHud: TMPro.TextMeshProUGUI
+Headers: List<GameObject>
+UpdateSalesmanRelationshipText()
+UpdateSalesmanMoney()
+UpdateNpcInventory()
+UpdateInventoryAndMoneyHud()
+DayOne()
+UpdateDay()
+EnableOrDisableExtendedHud()
WorldState
+ instance:WorldState
+ NpcPrefabs: List<GameObject>
+ SalesmanPrefabs: List<GameObject>
+ NpcGlobalList: List<GameObject>
+ SalesmanGlobalList: List<GameObject>
+ Day: int
- Start()
- Update()
- DayOne()
- NextDay()
- ActivateNpcWithNecesities()
- DesactivaAllNpc()
+ TestIfPassToNextDay()
Salesman
+money: float
+buySell: BuySell
+characterName: WorldState.SalesmanCharacterName
-shopState: ShopState
-BuyQueue: List<GameObject>
-TranformsQueuePosition: List<Transform>
-DaysInRedNumbers: int
-Awake()
+DayOne()
+NextDay()
+GetShopState(): ShopState
+IsQueueFull(): bool
+InsertInQueueLastPosition(GameObject npc): Transform
+UpdateQueue()
+GetShopExitPoint(): Transform
+GetLastPositionInQueue(): int
SalesmanBehaviour
-minMoneyToBuy: float
-minMoneyToBuyFrigdeAndWasher: float
-maxPercentageDifference: float
+ResupplyShop(float money, ref ShopInventory inventory, WorldState.SalesmanCharacterName salesman) : float
BuyBehaviour
+shopsChecked: List<WorldState.SalesmanCharacterName>
+shopToBuy: List<WorldState.SalesmanCharacterName>
+itemToBuyType: List<Item.ItemType>
+positionInQueue: int
+buyPreference: BuyPreference
+priceRelevance: PriceRelevance
+buyState: BuyState
-waitTimeToBuy: const float = 4
-probabilityToSeeAnotherShop: const float = 35
-coroutineStarted: bool
-OnEnable()
-GoToSeeShop()
-MakeBuyDecision() : bool
-GetBetterSalesman() : WorldState.SalesmanCharacterName
+GetPreferredSalesman() : WorldState.SalesmanCharacterName //for hud
-GoToBuy()
-GoToQueue(WorldState.SalesmanCharacterName salesman)
+SetQueuePosition(int queuePos)
+CheckInPosition()
+EnteredInQueue()
-BuyShop(): IEnumerator
-SeeShop(): IEnumerator
-WaitToQueueDrop(): IEnumerator
+ShouldBuyFood(ref Inventory inventory, float money):bool
+ShouldBuyFood(ref Inventory inventory, float money, int estimatedBuy):bool
+ShouldBuyDetergent(ref Inventory inventory, float money):bool
+ShouldBuyDetergent(ref Inventory inventory, float money, int estimatedBuy): bool
+ShouldBuyFrigde(ref Inventory inventory, float money): bool
+ShouldBuyWasher(ref Inventory inventory, float money): bool
Npc
+ money: float
+ characterName: WorldState.NpcName
+ cleanClothes: byte
+ salesmanRelationships: Dictionary <WorldState.SalesmanCharacterName, float>
- moneyWinnedEveryDay: float
- Awake()
+ GetPersonality(): Personality.PersonalityType
+ DayOne()
+ NextDay()
+ GoToShop(): bool
+ GoToPosition(Transform position)
+ GetSalesmanRelationships(WorldState.SalesmanCharacterName name) : float
- TestIfStopped(): IEnumerator
- DesactivateIfIsInExit()
+ CheckIfEndedAll()
+ ChangeRelationshipsOnUsedItems(Item item, bool isBrokenItem, bool brokenNow)
+ ChangeRelationshipsOnUsedDetergent(bool goodResult, WorldState.SalesmanCharacterName salesman)
+ ChangeRelationshipsOnUsedFood(int qualityResult, WorldState.SalesmanCharacterName salesman)
- InitialiceDictionaries()
SocialBehaviour
+SomeConstants
+ GetInteractionModifiers(Personality.PersonalityType targetPersonalityType)
AICharacterControl (default Unity class)
+ agent: UnityEngine.AI.NavemeshAgent
+ character: ThirdPersonCharacter (default Unity Class)
+ target: Transform
- Start()
- Update()
+ SetTarget(Transform target)
Personality
npcPersonality; PersonalityType
Inventory
+UsableItems: List<Item>
+FridgeItem: Item
+Washer: Item
+lastWasher: Item
+lastFridge: Item
+money: float
+AddNewItem(Item item)
+UpdateInventory()
+UseFood(): Object[bool, int, WorldState.SalesmanCharacterName]
+CleanCloth(): Object[int, WorldState.SalesmanCharacterName]
+DayOne()
ShopInventory
+MaxItemQuality: float
+MinItemQuality: float
+ProfitMargin: float
+FridgeList: List<Item>
+WasherList: List<Item>
+FoodList: List<Item>
+DetergentList: List<Item>
- Awake()
+ AddFrigdeToStore(WorldState.SalesmanCharacterName buyedFrom)
+ AddWasherToStore(WorldState.SalesmanCharacterName buyedFrom)
+ AddFoodToStore(WorldState.SalesmanCharacterName buyedFrom)
+ AddDetergentToStore(WorldState.SalesmanCharacterName buyedFrom)
+ DayOne()
+ ItemBought(Item item)
Item
+itemType: ItemType
+itemValue: float
+itemQuality: float
+ItemSpectedLifeTime: int
+ItemLifeTime: int
+ItemBaseLifeTime: int
+boughtFrom: WorldState.SalesmanCharacterName
+Item(ItemType itemType, float itemValue, float itemQuality, int itemSpectedLifeTime, int itemBaseLifeTime, WorldState.SalesmanCharacterName boughtFrom)
+Item(ItemType itemType)
+NextDay()
VisibleItemsInShop
-foodItems: GameObject[]
-detergentItems: GameObject[]
-fridgeItems: GameObject[]
-washerItems: GameObject[]
+ ChangeFoodItemsDisplayed(bool add, int actualNumber)
+ChangeDetergentItemsDisplayed(bool add, int actualNumber)
+ChangeFridgeItemsDisplayed(bool add, int actualNumber)
+ChangeWasherItemsDisplayed(bool add, int actualNumber)
+DayOne()
-HideAllItems()
1
1
1
1 ... n
1
1...n
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
n
1
1
1
n

Figure 4.1.17: General diagram class

4.2 Results

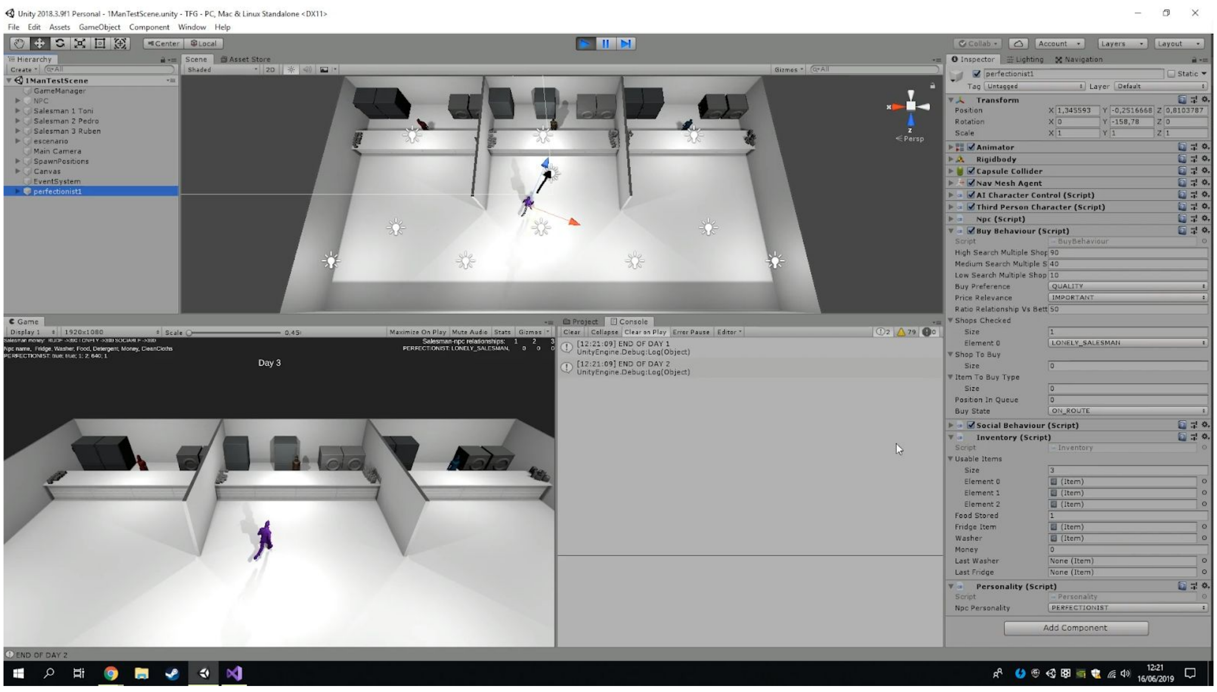
The results obtained from AI are those expected with the description on the Chapter 3.3. NPC tries to satisfy its needs buying first in his prefered salesman, and if not they will try to buy in the second prefered salesman, or the last.

NPC AI result to be non-linear, every time we run the game the results are different. We can not find two games that happens the same due to randomness of some aspects in the game, like the values generated on created items by the salesman, and the decisions of the NPCs.

In addition, the different behaviors of NPC and salesman can be adjusted just by touching their variables.

Now we will show the NPC in action, showing some screenshots and his decision flow. To do it more visible we will use a scene, called 1ManTestScene, when there is only one NPC with a perfectionist personality, who will visit all the shops. This is available in a video as well, link in the bibliography [9].

First the NPC will try to go to his prefered salesman.

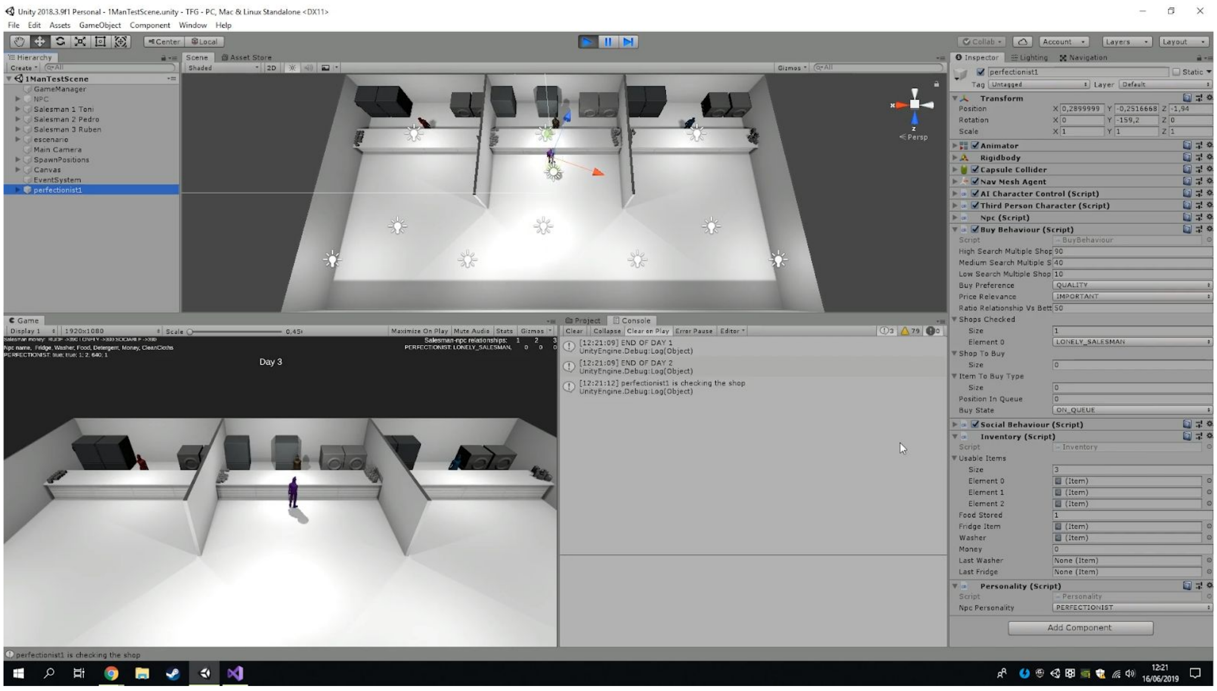


```
Salesman money: RUDE ->300 LONELY ->300 SOCIABLE ->300
Npc name, Fridge, Washer, Food, Detergent, Money, CleanCloths
PERFECTIONIST: true; true; 1; 2; 640; 1
Salesman-npc relationships:   1   2   3
PERFECTIONIST: LONELY_SALESMAN,   0   0   0

[12:21:09] END OF DAY 1
UnityEngine.Debug:Log(Object)
[12:21:09] END OF DAY 2
UnityEngine.Debug:Log(Object)
```

```
perfectionist1
Tag Untagged   Layer Default
Transform
Position   X 1,345593   Y -0,2516668   Z 0,8103787
Rotation   X 0          Y -158,78      Z 0
Scale      X 1          Y 1            Z 1
Animator
Rigidbody
Capsule Collider
Nav Mesh Agent
AI Character Control (Script)
Third Person Character (Script)
Npc (Script)
Buy Behaviour (Script)
Script                        BuyBehaviour
High Search Multiple Shop     90
Medium Search Multiple S      40
Low Search Multiple Shop      10
Buy Preference                QUALITY
Price Relevance               IMPORTANT
Ratio Relationship Vs Bett    50
Shops Checked
  Size                        1
  Element 0                   LONELY_SALESMAN
Shop To Buy
  Size                        0
Item To Buy Type
  Size                        0
Position In Queue             0
Buy State                     ON_ROUTE
Social Behaviour (Script)
Inventory (Script)
Script                        Inventory
Usable Items
  Size                        3
  Element 0                   (Item)
  Element 1                   (Item)
  Element 2                   (Item)
Food Stored                   1
Fridge Item                   (Item)
Washer                        (Item)
Money                         0
Last Washer                   None (Item)
Last Fridge                   None (Item)
Personality (Script)
Script                        Personality
Npc Personality               PERFECTIONIST
Add Component
END OF DAY 2
12:21
16/06/2019
```

Figure 4.2.1: NPC goes to first queue

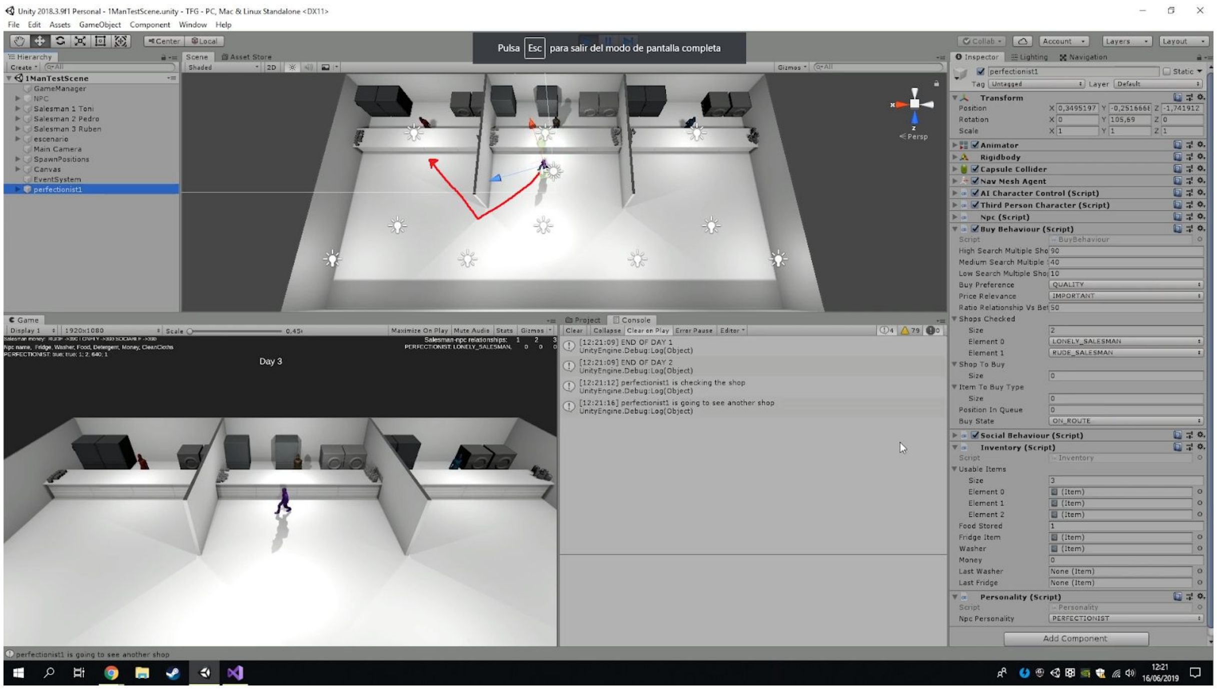
Then he will check the items available in the store.



```
[12:21:09] END OF DAY 1
UnityEngine.Debug:Log(Object)
[12:21:09] END OF DAY 2
UnityEngine.Debug:Log(Object)
[12:21:12] perfectionist1 is checking the shop
UnityEngine.Debug:Log(Object)
```

Figure 4.2.2: NPC see the items

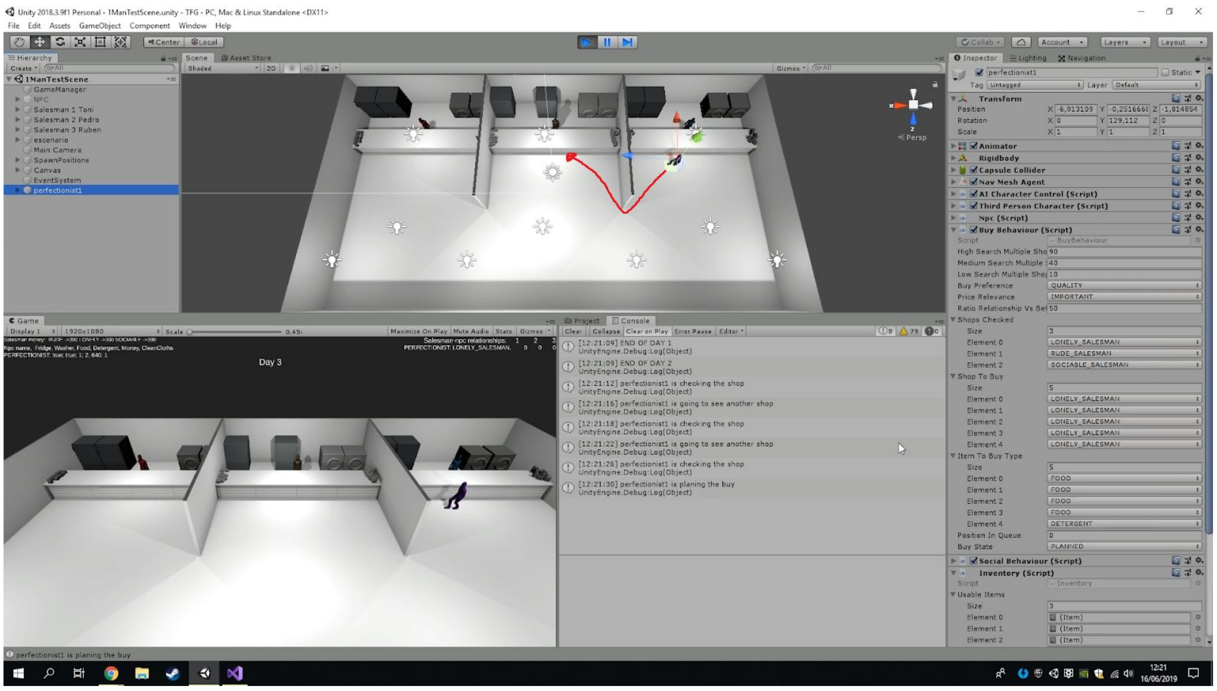
Before that, due to his personality, he will go to see another shop



```
[12:21:09] END OF DAY 1
UnityEngine.Debug:Log(Object)
[12:21:09] END OF DAY 2
UnityEngine.Debug:Log(Object)
[12:21:12] perfectionist1 is checking the shop
UnityEngine.Debug:Log(Object)
[12:21:16] perfectionist1 is going to see another shop
UnityEngine.Debug:Log(Object)
```

Figure 4.2.3: NPC go to see another shop

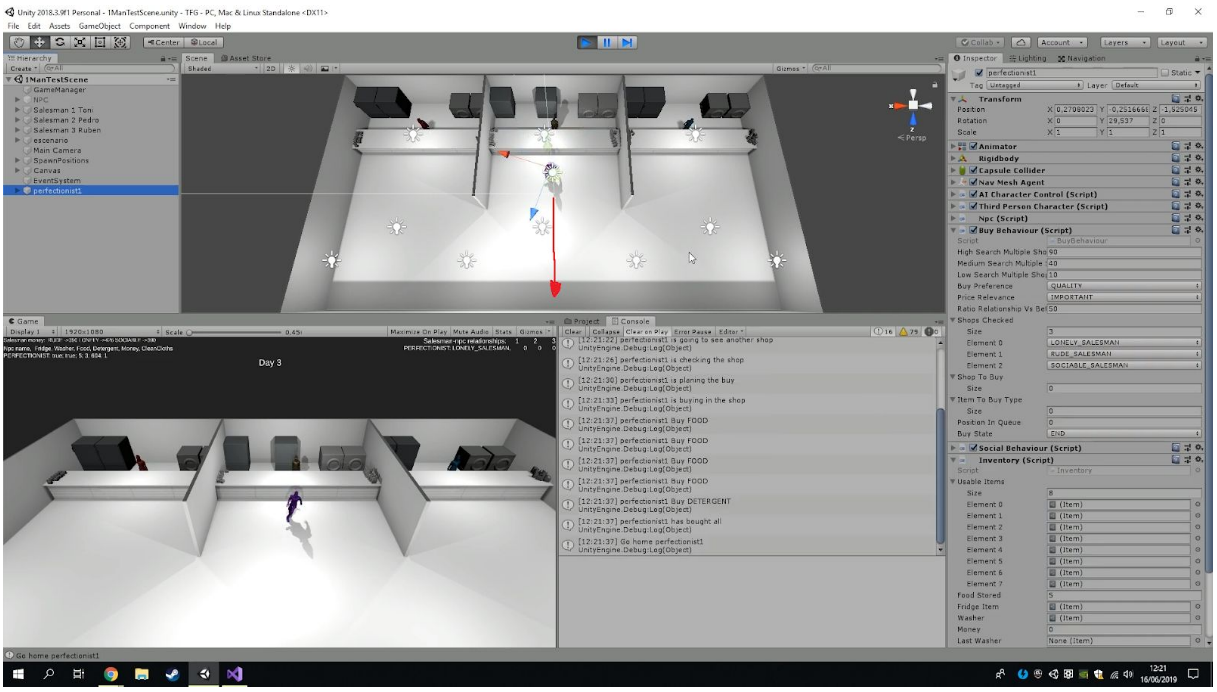
When he finishes seeing all the shops he will make the buy plan and will go to the designed store to buy the items. In this case the NPC will buy in the central shop, because he is his prefered salesman.



```
[12:21:09] END OF DAY 1
UnityEngine.Debug:Log(Object)
[12:21:09] END OF DAY 2
UnityEngine.Debug:Log(Object)
[12:21:12] perfectionist1 is checking the shop
UnityEngine.Debug:Log(Object)
[12:21:16] perfectionist1 is going to see another shop
UnityEngine.Debug:Log(Object)
[12:21:18] perfectionist1 is checking the shop
UnityEngine.Debug:Log(Object)
[12:21:22] perfectionist1 is going to see another shop
UnityEngine.Debug:Log(Object)
[12:21:26] perfectionist1 is checking the shop
UnityEngine.Debug:Log(Object)
[12:21:30] perfectionist1 is planing the buy
UnityEngine.Debug:Log(Object)
```

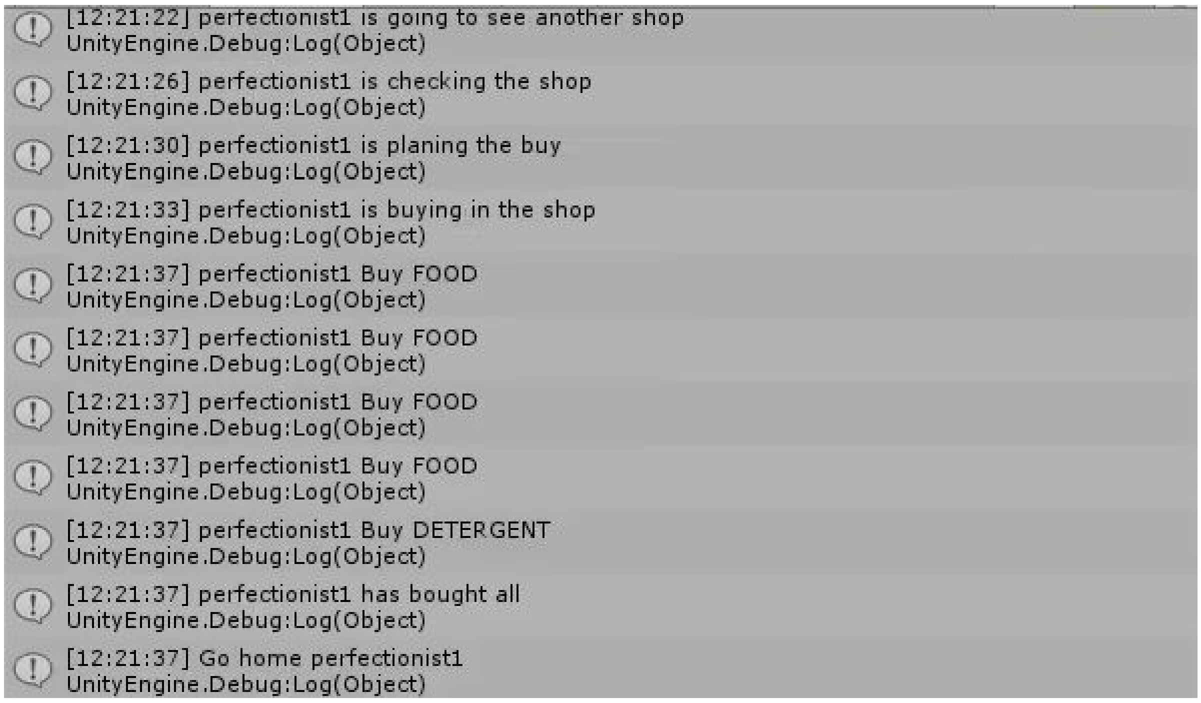
Figure 4.2.4: NPC makes the buy plan and go to buy the items.

When he finishes buying all items, he will returns home.



```
[12:21:22] perfectionist1 is going to see another shop
UnityEngine.Debug:Log(Object)
[12:21:26] perfectionist1 is checking the shop
UnityEngine.Debug:Log(Object)
[12:21:30] perfectionist1 is planing the buy
UnityEngine.Debug:Log(Object)
[12:21:33] perfectionist1 is buying in the shop
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy FOOD
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy FOOD
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy FOOD
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy FOOD
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy DETERGENT
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 has bought all
UnityEngine.Debug:Log(Object)
[12:21:37] Go home perfectionist1
UnityEngine.Debug:Log(Object)
```

Figure 4.2.5: NPC return home.



```
[12:21:22] perfectionist1 is going to see another shop
UnityEngine.Debug:Log(Object)
[12:21:26] perfectionist1 is checking the shop
UnityEngine.Debug:Log(Object)
[12:21:30] perfectionist1 is planing the buy
UnityEngine.Debug:Log(Object)
[12:21:33] perfectionist1 is buying in the shop
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy FOOD
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy FOOD
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy FOOD
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy FOOD
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 Buy DETERGENT
UnityEngine.Debug:Log(Object)
[12:21:37] perfectionist1 has bought all
UnityEngine.Debug:Log(Object)
[12:21:37] Go home perfectionist1
UnityEngine.Debug:Log(Object)
```

Figure 4.2.6: Debug of the last actions of the NPC before going home.

This is the work of a NPC in a general case. However, it is difficult to check how well AI will work in special cases. They must be forced to be able to visualize it well. To prove that it works correctly, we will modify some variable on the game in execution and then we will be able to see them easily. This special cases are demonstrated in the video that we commented before [9], after the general case.

This cases are the next:

Case 1, not enough money to buy all items in the preferred shop.
Reaction of the NPC: The NPC looks at all the stores and buys what it can in the favorite salesman and the object that can not buy, in the secondary salesman.

Case 2, not enough stock in shop.
Reaction of the NPC: The NPC looks at all the stores and buys in those preferred ones where there is stock.

Case 3, preferred salesman closed.
Reaction of the NPC: The NPC looks at the other stores according to personality, since it does not find its preferred quality.

Case 4, preferred salesman closes, and bad opinion from next salesman selected.
Reaction of the NPC: The NPC has obtained a bad result in the last purchase with the dependent on the left, so it is ready to buy in the other store because he has better relationship.

5

CHAPTER

Conclusions and future work

5.1 Conclusions

Now let's talk a little about the conclusions of this project. In general terms, the objectives have been accomplished, a non linear AI has been achieved, which do not seems to do the same all the time or being cyclic. As well, AI is capable of handling itself without errors and autonomously. In addition, it is easy to add more functionalities or incorporate new behaviors in a relatively simple way. This includes the addition of GOAP AI manager, which can be incorporated directly to the project.

However, on a personal level I am not satisfied with the performance of the AI, at least in terms of complexity, because one of the important parts of the project, the creation of a system of relations between the NPCs, which would give to the project a great complexity, was discarded in the middle of the development.

This idea will make that the relations between the NPC can modify the NPC relationship with the salesman. As well, with this system, the salesman could talk about the other salesman trying to convince the NPC to not buy to the other salesman. This idea was ready to be implemented, although it was not realized due to the lack of time, so I prefered to focus on finishing the entire store behavior without any bugs.

Even so, AI does its job well, without any bug, and has the advantage of being able to add or change things in the future.

5.2 Future work

To continue the project, several improvements or additions can be proposed to make it more complete.

The first is the inclusion of the NPC interaction system that we discussed on Point 5.1, adding new behaviors, such as a NPC that avoids queues, or incorporate brands of appliances, food, etc. Thus, in addition to select a salesman, the NPC could also select the brand, then the quality would not be linked to the salesman but to the brands offered.

Another thing that has to be done without a doubt is the change of the movement system, since the basic of Unity is very simple and often clumsy.

As well making some little changes in the code would be great, such as changing some functions or variables position or including a single script with all the game constants because they are distributed by several classes and sometimes it is annoying.

And finally, it would be interesting to introduce an AI that tries to maximize the income to the salesman, so then will be a real competition between them for maximum profit.

Bibliography

[1] Saw on 30/06/2019. https://www.wired.com/2015/12/open-world-games-2015/

[2] Saw on 30/06/2019.
https://www.theguardian.com/technology/2016/oct/10/mafia-3-review-1960s-shooter-gameplay

[3]Saw on 30/06/2019. https://www.youtube.com/watch?v=JZokdmzvDGA

[4]Saw on 30/06/2019. https://unity.com/es

[5]Saw on 30/06/2019.https://www.blender.org/

[6]Saw on 30/06/2019.http://lorca.act.uji.es/curso/latex/ejemplo/memoria.pdf

[7]Saw on 30/06/2019. https://github.com/franstuka/TFG-rep/tree/master/Docs/Gantt%20Diagram

[8]Saw on 30/06/2019. https://github.com/franstuka/TFG-rep/tree/master/Docs/Class%20Diagram

[9]Saw on 30/06/2019. https://youtu.be/_1rqW4NIM6A

[10]Saw on 30/06/2019. Source code: https://github.com/franstuka/TFG-rep

[11]Saw on 30/06/2019. Game Running in final scene: https://youtu.be/Og5t2kb8JNg