Bringing life into Open World Cities

How to populate cityscape environments with “Non-Playable Characters” pedestrians for open world games.

Master’s degree « jeux et médias interactifs numériques » cohabilited by the *Conservatoire national des arts et métiers et l’Université de Poitiers*, Angoulême

BADANA Tristan,
tristan.badana@gmail.com,
CNAM-ENJMIN

July-August 2022
Tutors : LEVIEUX Guillaume & BUENDIA Axel

ABSTRACT

This article aims to present and combine different methods for populating video game cityscapes environment with NPCs. This paper specifically focuses on creating pedestrians who interact with their environment and with each other, as well as how to dynamically spawn them.

CCS CONCEPTS

• **Computing methodologies** → *Artificial intelligence* → *Distributed artificial intelligence* → *Multi-agent systems*

KEYWORDS

video game, NPC, open world, city, population, immersion, smart objects, behavior, spawning

1 INTRODUCTION

Open world games are designed to immerse players into vast worlds they can explore and play with. To create lively urban cities, they are populated with pedestrians Non-Playable Characters (NPCs) which explore and interact with the game's environment.
However, manually placing and programming each NPC individually is both time and resource consuming. To address this, the general approach is to make their logic as systemic as possible and generate their placement procedurally.

1.1 Convincing NPCs for video games

First, it is essential to define what are convincing NPCs for video games.
This paper focuses on presenting and combining different methods to populate cityscape environments with pedestrians for open world games like *GTA V*, *Cyberpunk 2077* or *The Witcher*. In these games, they dynamically spawn, roam the city, interact with their environment and interact with each other.

The subject being vast and tightly bound to game design, we won't be able to cover every aspect of the implementation or go too deeply into certain details. This paper focuses on giving a flexible base which can be iterated and optimized for similar games.
For this reason, we won't or briefly be talking about vehicles and traffic navigation, online constraints, reaction to player's actions, tooling and crowd simulation.
These aspects remain important for creating immersive open world games, and this proposition aims to be compatible with their implementation.

2 INTERACTIONS WITH THE ENVIRONMENT

The first step to making a faithful representation of pedestrians is to have them interact with their environment. People generally perform a variety of activities when they are outside, such as sitting on a bench to rest, stopping at a store window or retrieving money at an ATM.

Each object needs to have a specific behavior the NPC can perform after going toward it.
But putting each behavior into the pedestrians' logic can complicate future features' development. To prevent this and increase flexibility, these behaviors can be delegated to the objects in the environment instead.
This approach was notably explored in *The Sims* where the AI interacts with "Smart Objects" to perform actions like cooking, reading a book and more.

2.1 Smart Objects

Smart objects (SO) are static assets in a scene which carry information about how they can be used. For instance, a bed can carry information on how an agent can sleep on it. It even provides the necessary animation data for doing so. This allows the creation of ambient behaviors without altering the AI of the NPCs.

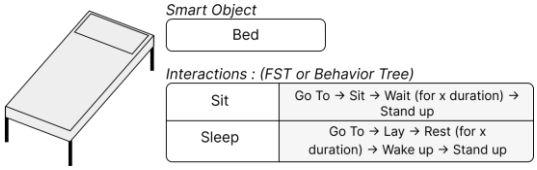

Figure 1. Example of a Smart Object with its different interactions.

Smart Objects control NPCs' interactions using a Finite State Machine (FSM) or a Behavior Tree (BT). They mostly aim to play a sequence of animations. Once it is done, the agent goes back to its previous activities.

There are two approaches to having NPCs use SO: either NPCs seek out nearby Smart Objects to interact with, or the Smart Objects attract NPCs.
In open world games, the latter is more effective because we want many people to populate our city while keeping their logic decoupled from Smart Objects' one; If a Smart Object is not already in use, it periodically emits signals to nearby pedestrian and have a chance of attracting one.

2.2 Additional features

Smart Objects are very flexible and have a great potential. They can be used in numerous ways to adapt to different kinds of open world games and bring as much life as possible. Below are some commonly implemented additional features :

- **Invisible Smart Objects**: Smart Objects do not need to be visible; they can be locations where an agent performs specific animations. For example, an invisible SO could be placed near a monument where NPCs could take photos with their phone.
- **Varied interactions**: Putting different interactions and animation sequences on one SO adds variety and personality to their users.
- **Smart Objects for specific NPCs**: to bring diversity to the world, some Smart Objects can be designed for specific types of NPCs. Meaning only tourists would use a monument SO for example. This feature implies modifying NPCs by giving them at least an occupation attribute.

There are many more additional features which can be added on top of its basic implementation. This proves they are a great addition to open world games, and this is why they frequently used in this genre. For more information about their implementation, refer the Appendix 1.

3 INTERACTIONS WITH OTHER NPCS

The second aspect of making ambient life more believable in a cityscape environment is to make NPCs engage with each other. This is important because it creates group dynamics and allows for more complex interactions.

Based on the idea of delegated logic, Smart Locations take inspiration from Smart Object but focus on NPC-to-NPC interactions.

3.1 Smart Location

"Smart Location" (SL), or "Smart zones" are similar to Smart Objects : they also carry information about how they can be used and provide the necessary animation data for doing so, allowing to create ambient behaviors without modifying the AI of the citizens. But instead of representing 1 object which can be used by 1 NPC, they are invisible objects that refer to multiple concrete objects.

They also differ on how their interactions – referred as actions – are triggered. For that they use 3 elements:

- **Roles**: Define which kind of actors can participate in a script and how many. These are dynamically attributed as a participant joins the Smart Location. For example, there could be 3 agents with the role customer, and 1 with the role waiter.
- **Rules**: Indicate what actions the actors can perform based on prequalification. For instance, if there are at least 2 Customers at a restaurant table, one will Talk and the other Listen. If there is at least 1 Waiter and 3 Customers, one of them will place an Order.
- **Blackboard / Tuple Space**: Serves as a shared space to store and access information.

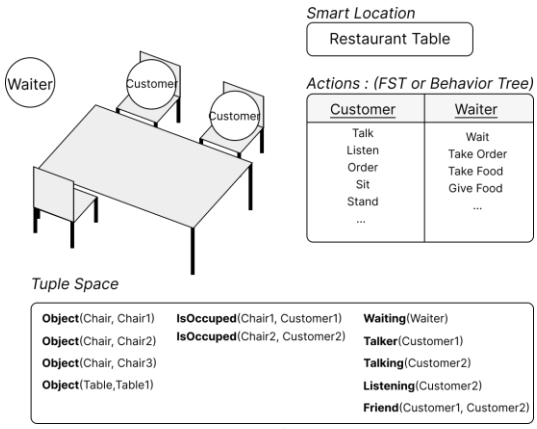

Figure 2. Example of a Smart Location "Restaurant Table" with the different roles, actions and an example of Tuple Space.

Every interval of time, Smart Locations inspect each NPC and check if they validate the prequalification of each rule based on their role and the blackboard. If they do, the inspected NPC perform the action bound to the rule.

A particularly advanced implementation of Smart Location can be found in *Final Fantasy XV*[1] where developers used a Tuple Space instead of a Blackboard for multiple reasons such as tooling and gain of time for programmers. For more information about its implementation, you can read the Appendix 2.

3.2 Smart Objects or Smart Locations

Like Smart Objects, Smart Locations delegate interactions with environment but also support NPC-to-NPC interactions. Moreover, all additional features discussed in "2.2 Additional features" apply to SL. But should Smart Objects and Smart Locations coexist in the same game?

Implementing both could result in implementing features twice. But separating them makes for a clearer level design and a better behavior management. Furthermore, some games allow the player to use SO

like in *The Sims*. Therefore, the decision to use one or both is to be considered and depends on the game.

3.3 NPC Groups

The second method for allowing NPCs to interact with each other is by creating grouped behaviors.
The chosen group system takes inspiration from the Smart Location concept. A group is an invisible game object dynamically instantiated. Like SL, a group has assigned roles : 1 Leader and up to 4 Followers, forming groups of 2 to 5 individuals. These roles are stored in a blackboard bound to the group.
The Leader decides who is talking by picking a random person in the group. Meanwhile, the blackboard will store who is the current talker and play a talking animation for them, all the other group members will look at the current talker and play a listening animation.
If the Leader is unable to lead anymore (due to death or despawning), another Leader is chosen among the members of the group.

Finally, there are 2 ways of forming groups. Firstly, when NPCs are using an SL, they are grouped with others sharing the same role.
The second way is to spawn NPCs near each other and create a group. More details at 5. SPAWNING NPC.

Combining SL and NPC Groups allows for simple interactions while walking and more complex ones when using objects in their environment.

4 BEHAVIOR

4.1 Navigation

Pedestrian navigation typically involves them roaming the cityscape. To simplify and optimize their movements, a simple navigation graph is sufficient.
First, setting up a Navmash allows pedestrians to walk while avoiding obstacles. Placing nodes on the NavMesh along the roads or sidewalks and linking them creates lanes the NPCs will be able to walk by.

Figure 3. Lanes used in Assassin's creed Odyssey. GDC Conference : Virtual Insanity. Via YouTube (https://www.youtube.com/watch?v=a09vnDjmY_E)

Finally, to have NPCs roam in the city, they are given a far Node as a destination when they spawn and use the A* algorithm[2] to navigate through these nodes. If the lanes are not obstructed, a navmesh isn't required for the roaming navigation. However, since they may need to perform other behaviors (for instance : flee from danger), it is always practical to set up one.

To enhance realism, some features can be added. To begin with, nodes can be given a width. By applying a random angle and random distance offset to each NPCs on the position of the nodes (based on its width and the direction of the lane), wider lanes are created. This enables NPCs to walk side by side and prevent them from all walking in the center of the road. Moreover, implementing Reciprocal Velocity Obstacles (RVO)[3] helps them dynamically avoid each other.

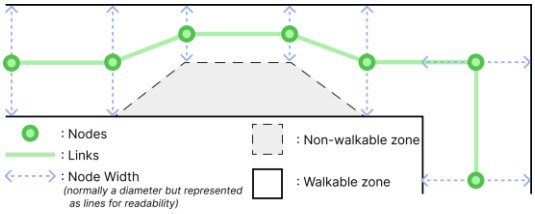

Figure 4. Schematic drawing of a lane using node width to create large walkable areas.

Finally, for crossroads, the A* algorithm can be modified to support temporarily disabled nodes. If the next target node of an NPC is disabled, it waits at its current node until it's re-enabled.

When an NPC is not part of a group or using a SO/SL, it enters a roaming state. During it, it looks for a relatively distant node in the road navigation graph and go toward it as said earlier.
For groups, only the Leader navigates like standard pedestrians. Followers walk beside it, using its position and a calculated offset. To prevent followers from walking on the road, this offset is dynamically calculated based on the width of the next node in the Leader's path, similarly to how the random offset was determined for navigation. The formula is as follows:
posFollower =
*posLeader + clamp (offset * LeadNextNodeWidth/2 ,-maxDist, maxDist)*

4.2 Behavior transition

With this implementation, the NPCs have 3 types of behaviors: Solo, Grouped and Delegated (for Smart Objects and Locations).

The solo and grouped behavior discussed here are relatively simple. However, depending on the game, different type of groups, or more complex solo behavior

may be more suited, such as displaying emotions, or reacting in different ways to the player actions.
A simple and flexible solution is to use an FSM or a BT for solo behavior, rather than separate FSMs for each of the three behavior types. This FSM deactivates when the NPC joins a group or interacts with a SO/SL and is reactivated by the object once its delegated logic ends.

On top of this, giving each NPCs a local blackboard which stores references to which group they belong to or which SO/SL they are using can allow to send signals when necessary.

5 NPC SPAWNING

NPC spawning in open world games is crucial to immersion, but several constraints must be considered :

- **Performances**: The player should feel like the NPCs are always roaming the city, but due to performance issues, pedestrians cannot be on the map at every moment of the game.
- **Visibility**: As such, the spawning should be hidden from the player, as seeing the NPCs appear would break the immersion.
- **Realism**: Pedestrian should spawn at logical locations or areas people typically walk like sidewalks, avoiding lanes where cars ride.
- **Density**: Population density should vary based on location; a shopping district is more crowded than an industrial one.

5.1 Spawning System

A solution which addresses most of the issues is to use a grid to spawn NPCs.

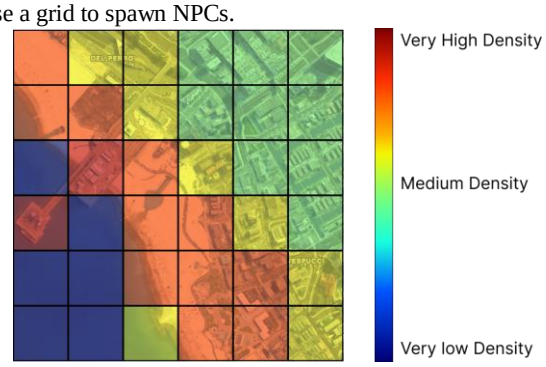

Figure 5. Example of grid using GTA V map.

This grid covers the entire city. Each tile needs to be at least as large as the chosen render distance of NPCs of the player's camera to ensure the spawning is hidden from the player.
Pedestrians are spawned in the tile the player is currently in and its adjacent tiles (9 tiles in total). Using this system, NPCs are only present near the player.
To optimize further, instead of spawning NPCs whenever the player changes tile, a collider is put around the tile the player is currently in, called the "Respawn Trigger".
When the player reaches the edge of the Respawn Trigger, NPCs are spawned in the tile the player is currently in and the adjacent ones (if NPCs are not already spawned), the NPCs in the previously adjacent tiles are despawned. The Respawn Trigger then moves to the center of the player's current tile. For more details on the Respawn Trigger, refer to Appendix 3.

To ensure that pedestrians spawn on the sidewalks and roads they are supposed to walk, they are spawned around the nodes of the roads used for their navigation. Using the width of the nodes to prevent them all walking in the center of the road.
Since pedestrians begin walking before the player sees them clearly, their exact spawn positions are difficult to notice.

Moreover, assigning each tile a specified number of NPCs they should spawn helps control density.
To create a realistic ambient life NPCs should be spawned on Smart Objects and Smart Locations in priority. This involves providing these objects instantiating instruction and filling their blackboards at specific action steps. Spawning on SOs also allows to spawn agents on unreachable places such as terraces or building interiors.

Furthermore, groups can be spawned by placing multiple NPCs at the same node and create a group object to represent them.
Additionally, spawners activated by the Respawn Trigger can be placed to spawn NPCs. Lastly, to prevent heavy spawning cost, using pooling avoids repetitive instantiations.

With this implementation, pedestrians are only present around the player at a maximum of twice their render distance. They only despawn when they are too far from the player and spawn at coherent locations.

5.2 Context spawning

A common feature in open world games is context-based spawning. For example, policemen or firetruck may spawn due to player actions or the state of the world. A grid system supports this well as it can spawn these NPCs at adjacent tiles. However, it is important maintain game balance, this can be done by fixing a limit of enemies or type of NPCs which can be present simultaneously.
This approach was chosen for *Saints Row 4*.

6 OPTIMIZATIONS

To increase the number of NPCs displayed on screen, Levels of Detail (LOD) is an effective solution, not just for visuals but also for logic.
In *Assassins Creed Unity*[4], a 3-tier LOD is used for NPCs based on distance. The closest ones have their full logic, animation and their detailed model. The ones in the second LOD tier have fewer bones, simpler AI, limited 3d models and less complex animations. Finally, the furthest pedestrians have even more simplified AI, models, bones, but also a simplified collision system.

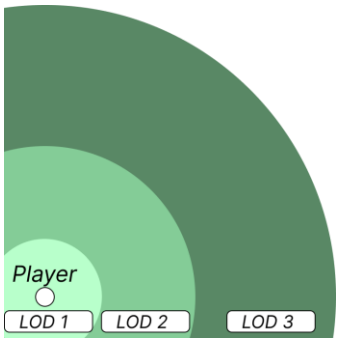

Figure 6. Schematic drawing representing different LOD tiers based on distance.

Another optimization consists of simplifying NPCs' behavior based on how long the player last saw them.

With what was presented in this paper, these two solutions can be applied by disabling RVO few seconds after an NPC is out of sight when it's in the second level of LOD. At the third level, their physic can be simplified by allowing pedestrians to walk through each other and looping the current actions in SO/SL and groups.
Implementing this requires careful handling of transitions between LODs, notably on the AI transition and the bone retargeting system.

7 CONCLUSION

With this implementation, pedestrians dynamically spawn as the player explores the city. They can be seen walking along the sidewalks or interacting with objects in their environment. Some will form groups in which they interact with each other.

Of course, some aspects of populating cityscape environments can be simplified and depend on the available budget. Different games prioritize specific features, like *Watch Dogs 2* which emphasizes NPC interactions that create emergent situations, while *Final Fantasy XV* centers on a variety of environmental interactions.

As mentioned in the introduction, there are still several aspects of populating open worlds which were not covered in this paper. Cost wise, NPCs require a significant amount of 3D assets, sounds and animations. Effective tools are also crucial as they give Level Designers more control and can save much development time. Also, more optimizations can be made and adapted to the game but should consider possible online constraints.

Many of these concepts have already been implemented in existing games. For instance, Smart Locations are utilized in *Final Fantasy XV*, the lane system is part of the Mass plugin for Unreal Engine and *Assassin's Creed Odyssey* also employed a similar system. The Spawning System and the Group Behavior are exceptions as the first one draws inspiration from Unreal Engine's world partitioning system and the second from *Assassin's Creed Odyssey's* groups.
While it would be valuable to combine all these concepts into one project to validate the ideas presented in this paper, doing so would require significant time and resources. This is why no implementation is provided here.

References

[1] Hendrik Skubch, 2015, *Ambient Interactions Improving Believability by Leveraging Rule-Based AI*, Retrieved July 25, 2024 from https://www.gameaipro.com/GameAIPro3/GameAIPro3_Chapter35_Ambient_Interactions_Improving_Believability_by_Leveraging_Rule-Based_AI.pdf

[2] Wikipedia, *A* search algorithm*, Retrieved June 21, 2024, from https://en.wikipedia.org/wiki/A*_search_algorithm

[3] Markus Buchholz, 2023, *Reciprocal Velocity Obstacles (RVO) for collision avoidance in C++*, Retrieved June 21, 2024, from https://markus-x-buchholz.medium.com/reciprocal-velocity-obstacles-rvo-for-collision-avoidance-in-c-d7f4e7959417

[4] Ubisoft Francois Cournoyer, 2015, *Massive Crowd on Assassin's Creed Unity: AI Recycling*. Retrieved July 24, 2024 from https://www.youtube.com/watch?v=Rz2cNWVLncI

Resources

For more precisions on certain subjects, you can consult these external resources which were also used for writing this paper. Some appear in multiple categories since they discuss multiple subjects.

Spawning

- *Living City in Mafia 2* : GDC session, 2010, Jan Kratochvvil, https://gdcvault.com/play/1013704/Living-City-in-Mafia
- *Virtual Insanity: Meta AI on Assassin's Creed: Origins* : GDC session, 2018, Charles Lefebvre, https://www.youtube.com/watch?v=a09vnDjmY_E
- *Free Range AI: Creating Compelling Characters for Open World Games* : GDC session, 2014, Jeet Shroff and Aaron Canary https://www.youtube.com/watch?v=jDCFMITrtHc

Smart Objects

- *Knowledge is Power: An Overview of Knowledge Representation in Game AI* : GDC Session, 2018, Daniel Brewer and Rez Graham, https://www.youtube.com/watch?v=Z6oZnDIgio4
- *Not Just Planning STRIPs for Ambient NPC Interactions in Final Fantasy XV* : Nucl.ai 2015 conference, 2015, Hendrik Skubch, https://www.youtube.com/watch?v=LwuJekXTozo
- *Ambient Interactions : Improving Believability by Leveraging Rule-Based AI*, Game AI Pro 3, 2017, Hendrik Skubch, https://www.gameaipro.com/GameAIPro3/GameAIPro3_Chapter35_Ambient_Interactions_Improving_Believability_by_Leveraging_Rule-Based_AI.pdf
- *Helping It All Emerge: Managing Crowd AI in Watch Dogs 2* : GDC Session, 2017, Roxanne Blouin-Payer, https://www.youtube.com/watch?v=LHEcpy4DjNc
- *Smart Zones to Create the Ambience of Life*, Game AI pro 2, 2015, Etienne de Sevin, Caroline Chopinaud, and Clodéric Mars, https://www.gameaipro.com/GameAIPro2/GameAIPro2_Chapter11_Smart_Zones_to_Create_the_Ambience_of_Life.pdf

Navigation

- *Virtual Insanity: Meta AI on Assassin's Creed: Origins* : GDC session, 2018, Charles Lefebvre, https://www.youtube.com/watch?v=a09vnDjmY_E

NPCs' Behavior

- *Helping It All Emerge: Managing Crowd AI in Watch Dogs 2* : GDC Session, 2017, Roxanne Blouin-Payer, https://www.youtube.com/watch?v=LHEcpy4DjNc
- *Not Just Planning STRIPs for Ambient NPC Interactions in Final Fantasy XV* : Nucl.ai 2015 conference, 2015, Hendrick Skubch, https://www.youtube.com/watch?v=LwuJekXTozo
- *AI Summit: Branching Out: 'Watch Dogs Legion's' Architecture for Group AI Behaviours* : GDC Session, 2021, Christopher Dragert and Patrick McKenna, https://gdcvault.com/play/1027239/AI-Summit-Branching-Out-Watch

Cars

- *AI Summit: Cities at Scale: Simulating Street Life on a Budge* : GDC session, 2022, Sandy MacPherson, https://gdcvault.com/play/1027985/AI-Summit-Cities-at-Scale
- *Living City in Mafia 2* : GDC session, 2010, Jan Kratochvvil, https://gdcvault.com/play/1013704/Living-City-in-Mafia

Appendices

A1 SmartObject implementation

Below is a simplified implementation of Smart Objects for a very basic context, it aims to clarify how they work.

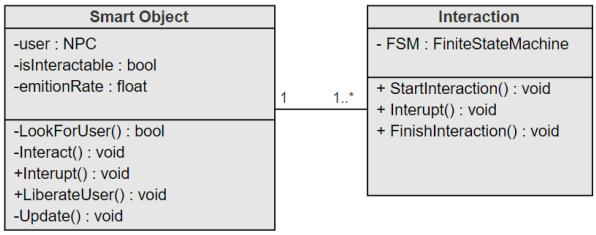

Figure 7. Simple class diagram of Smart Object (doesn't take tooling into account)

A `SmartObject` script can be attached to a GameObject in a scene. It has 2 main public fields:

- **emitionRate**: control the frequency of signal emissions to find users,
- **interaction**: a list of Interaction the NPC can perform.

When the `LookForUser()` function finds a user one of the interactions is triggered randomly. The FSM manages a sequence of animations, with each animation represented as a state.

Using an FSM facilitates transition from states to states and allows for dynamic switching between Interactions. This is notably useful for Smart Locations as multiple Actions may start at a same state. Alternatively, this can also be achieved with a Behavior Tree (BT).

The `Interupt()` functions are essential for reacting to the player's actions, however, the FSM should include one or more interruption states.

A2 SmartLocation implementation

Below is a simplified implementation of Smart Locations for a very basic context, it aims to clarify how they work.

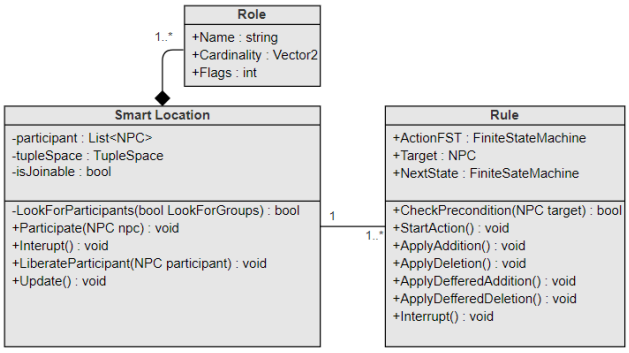

Figure 8. Simple class diagram of Smart Location (doesn't take tooling into account)

A `SmartLocation` script can be attached to a GameObject in a scene. It has 2 main public fields:

- **emitionRate**: control the frequency of signal emissions to find users/participants,
- **rules**: a list of Rule the NPC will check. If the preconditions are met, the corresponding action will be triggered.

The `Update()` function evaluates the preconditions for each rule for each participant by calling `CheckPrecondition(target)`. If the target fulfills the preconditions, `StartAction()` starts the action associated with the rule.

Similar to Smart Objects, the actions are FSM, stored in the rules here.

Actions may modify the Blackboard or the Tuple Space, the `ApplyAddition()` `ApplyDeletion()` etc… aim to facilitate the updates on the Blackboard and Tuple Space.

Tuple Spaces are particularly interesting for Smart Locations as basic operations can be implemented to facilitate the manipulation of data. The most useful are insertion, removal and queries used for checking preconditions. These are very useful for Tooling and gain time.

A tuple space can be thought of as a multimap, where tuples names are used as keys. For Example: the **Object** Tuple structure is ("Object", "type", ObjectReference). ("Object", "Chair", Chair1) is one of the Objects in the following figure, or **Object**(Chair,Chair1) for clarity.

Tuple Space

Object(Chair, Chair1)	**IsOccuped**(Chair1, Customer1)	**Waiting**(Waiter)
Object(Chair, Chair2)	**IsOccuped**(Chair2, Customer2)	**Talker**(Customer1)
Object(Chair, Chair3)		**Talking**(Customer2)
Object(Table,Table1)		**Listening**(Customer2)
		Friend(Customer1, Customer2)

Blackboard

Object :		**IsOccupied :**		**Waiting :**	**Talker :**	
Type	*Name*	*Chair*	*Customer*	*Waiter*	*Customer*	
Chair	Chair1	Chair1	Customer1	Waiter1	Customer1	...
Chair	Chair2	Chair1	Customer1			
Chair	Chair3					
Table	Table1					

Figure 9. Comparison between data stored in a Tuple Space vs typical Blackboard

This implementation is based *Final Fantasy XV*'s one. For more information, please consult their conferences on the subject or read their article on Game AI Pro.

A3 Respawn Trigger

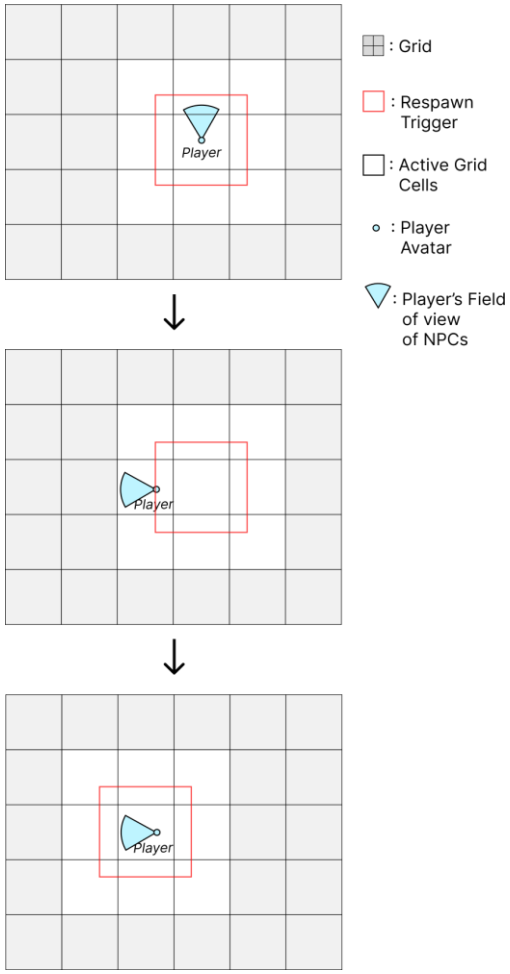

Figure 10. Schematic drawing illustrating how the grids are activated as the player moves.

The Respawn Trigger size can be mathematically calculated based on the size of the cells using this formula :

triggerRespawnSize =
*gridCellSize + ((gridCellSize*2 – NPCRenderDistance)*2)*