Social Deduction AI

Andrea Tassi

September 3, 2024

Contents

Abstract

The aim of this project is to design and implement an AI agent in Unity that mimicks the behaviour of a human player in a typical social deduction game.

In this case, more agents have to roam around the map gathering information, while one of them is a killer. If one of the agents find a dead body, a meeting starts, during which each agent share its relevant information, in order to identify a plausible suspect and vote it out.

Chapter 1

Project Overview

1.1 Project Aims

The aim of this project is to design and implement an AI agent in the Unity Engine that can mimick the behaviour of a human player in a typical social deduction game.

1.2 Inspirations

For this project, the structure has been inspired by already existing video games of the same genre, mostly *Among Us*, but changed accordingly to the aims, the needs and the limitations imposed by this project. Some assumptions have been made, in order to make it not too complex, thus functional for its aims. These assumptions will be discussed when needed.

1.3 Game Structure

The game flow is simple and can be compared to a single turn of *Among Us*, from the start to the end of the first "Emergency Meeting", when this project scope ends.

In particular, some agents have to roam around the map gathering information about the others through their field of view, while one of them is a killer, which can kill one of the others. If one of the agents find a dead body, a meeting starts, during which each agent share its relevant information, based on the data already available to the group, and making simple logical inferences, in order to identify a plausible suspect and vote it out of the group.

If the innocent agents can spot the killer, they win, otherwise the killer will be the winner.

1.4 Project Structure

The project consists mainly of two parts:

1. **Roaming**, in which the agents roam around the map, while doing tasks and gathering information.
2. **Meeting**, in which the agents share their information to identify the killer.

The project is mainly focused on the gathering information aspect and on the meeting phase. The two phases, together with their design and implementation details, are described in the next chapters.

Chapter 2

Roaming

The Roaming phase consists of the agents roaming around the map in order to do random tasks at some points of interest (POIs) and retrieving data from their field of view (FOV).

The killer also has to kill one of the other agents while roaming around the map.

2.1 Map

The map is composed of:

- **Locations** (rooms, anterooms and corridors), each one with its identifier.
- **Walls** and **obstacles**, which were added to limit and test the FOV of the agents, in order to see how they would impact on the information retrieval.
- **POIs**, where agents must go to accomplish their tasks.
- **Spawn points**, where the agents spawn at the start of the roaming phase.

Figure 2.1 shows the map designed for this project.

2.2 Agent

`Agent` is the main component of the AI agent.

2.2.1 Design

At the start of the Roaming phase, all the agents receive some random tasks to accomplish at their corresponding POIs. The tasks are executed in order of proximity: the next task chosen is the closest one to the current agent position, by simply using euclidean distance. Each task has a time to get done, during which the agent stays still in position.

Once an agent completed all of its tasks, it requests another random set and goes on.

If an agent is killed, it must stop roaming, doing tasks and gathering information.

2.2.2 Implementation

The agent is implemented by the `Agent` component.

The pathfinding is implemented through a simple navigation mesh, automatically generated by the `NaveMesh` system provided by Unity. Since the corridors are small and the POIs are shared by all of the agents, they can compenetrate each other, in order to facilitate their movement (as in *Among Us*).

The agents save their current location by colliding with it.

They also detect if they are at their current task POI by colliding with it: this starts a `Coroutine` on the `Agent` component, which waits for the time that the task needs to be accomplished.

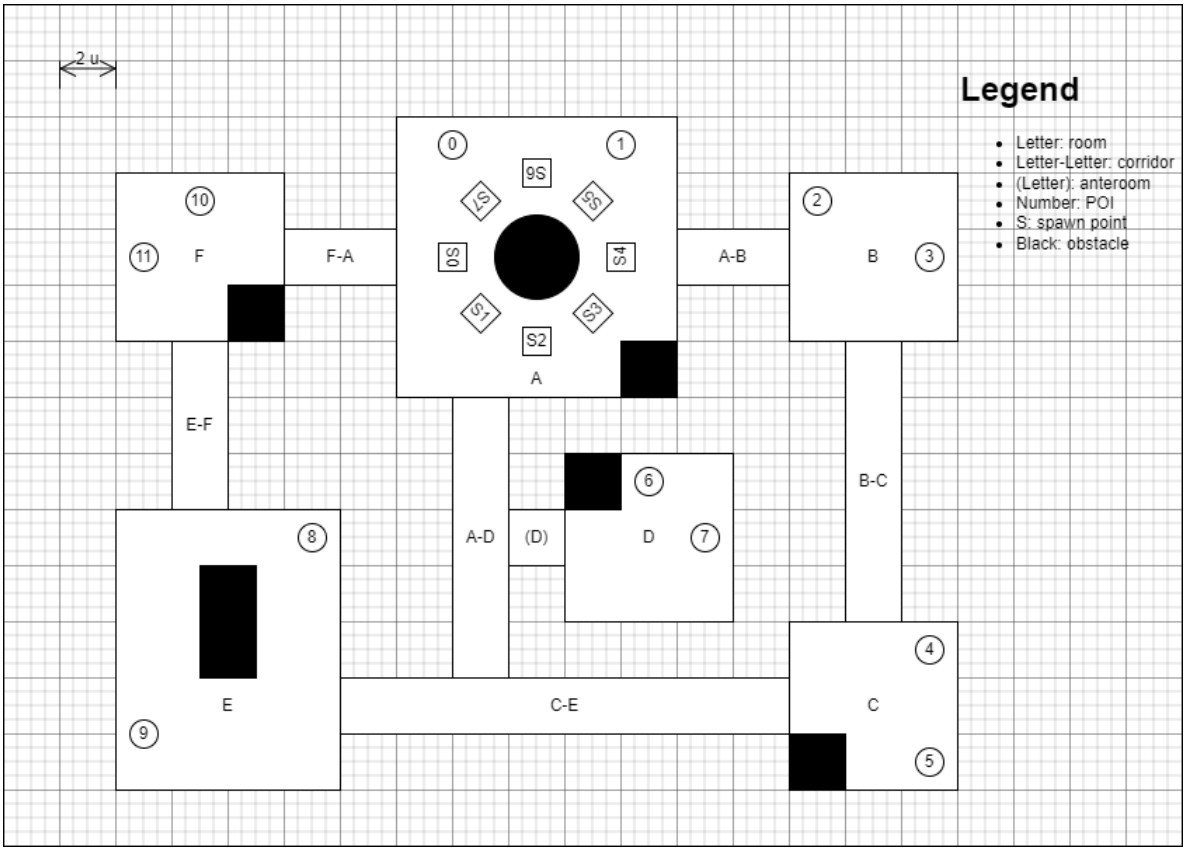

Figure 2.1: The map of the game scene.

2.3 Killer

`Killer` is the main component of the AI killer.

2.3.1 Design

The killer is also an agent, thus it needs to have the same behaviour as the other agents. Furthermore, it has another behaviour: it can kill an agent (after a cooldown) as soon as they enter in contact to each other.

A report cooldown is included as well, in order to avoid the killer to report a dead body instantly when it kills it, but leaving it the possibility to do so later on, when it will eventually encounter the dead body again before the other agents.

2.3.2 Implementation

The killer is implemented by the `Killer` component, which is a subclass of the `Agent` component.

The kill and the report cooldowns are implemented by `Coroutine`s on the `Killer` component, while the `Kill` action is implemented by detecting a collision with another agent.

2.4 Field of View

`FieldOfView` is the component that gathers information about the other agents while its owner is roaming around the map.

2.4.1 Design

The FOV of the agents has a cone structure, with a radius and a maximum angle, and its line of sight (LOS) can be blocked by obstacles, but not by other agents: this is done because they can also compenetrate, so they should be able to see everyone in a group if they gather together in a single spot.

If one of the other agents enters the FOV and can be seen by the LOS, its relevant information (agent, location and timestamp) will be gathered. If the same agent remains seamleassly visible (or if it is visible again in a short time), data will be updated and its timestamp will become a time interval, otherwise it will be a different information and has to be stored independently. This is done in order to avoid duplicated (or very similar) information.

If an agent sees a dead body, it saves this information and reports it to the others, starting a meeting.

2.4.2 Implementation

The radius check is done by an `OverlapSphere` with the specified radius, checking if there are some agents in range of sight.

Then, the angle check is done by comparing it to the angle between the forward direction and the direction of the vector connecting this agent to the ones in range.

Finally, the LOS check is done on the remaining agents by a `Raycast`.

Chapter 3

Meeting

When an agent finds a dead body, the meeting phase starts.

3.1 Setup

During the meeting setup, the agent that reported the dead body share this information to the others.

The other agents share their current position at the time of the report.

3.2 Rule-Based System

The reasoning phase is driven by a simple Rule-Based System (RBS).

During this turn-based phase, the agents can reason on the information they have in their database, making logical inferences and sharing their data about that (data that led to the conditions and data that emerged from the actions). The other agents store these new information in their own database, in order to be able to reason about them in their turn, by comparing and matching them with their own information or to other external information they may have acquired during the meeting.

The RBS snippets from the slides of the course have been used, modified accordingly with the needs of the project:

- Data is not stored as strings and literals, but as `Info` instances, so they can be queried more easily.
- Queries match more than one item in data, in order to filter it with the other queries.
- Running the RBS returns a boolean, in order to manage the case where no rule was fired (the agent simply cannot reason on the current insufficient data).

3.2.1 Rules

All the rules share a similar structure, with one condition and one action:

- The **condition** is composed by three queries. Each binding and match is stored internally for the duration of the turn, not only because they are used by the other queries of the condition, but also because they are used by the action, that needs to share them. The three queries can be classified in three different types:
  - **Binding query**, which retrieves the bindings for the next query (and for the action).
  - **Condition query**, which matches the searched data with the bindings provided before.
  - **Repetition query**, which avoids repetitions in the matches of a condition. This is needed because, in this RBS, the database changes only by adding new data, and not by modifying already present data, so the same matches can happen. Furthermore, data cannot be removed from the database, because it can be useful for the matches of other conditions. So, in this case, the adopted solution is to check if the data provided by a match is already present in the database.

- The **action** consists of four phases:
  - Inferring new data based on the condition bindings and matches.
  - Storing just inferred new data into the agent's database.
  - Adding some points to the level of suspect based on the action (some actions have negative points to decrease the level of suspect).
  - Sharing new data and matched data with the other agents. This can eventually lead to duplicate data in other agents' databases, as the matched data may come from other agents that shared it in previous turns during the meeting. However, these repetitions would be irrelevant, as only the first match fires, and, in any case, they would be avoided by the repetition query.

The rules implemented for this RBS, as illustrated in Table 3.1, are:

- **Agent close to dead body**: if an agent was close to the dead body (at a close time), then it's suspicious (as it may have killed it) and its suspect level increases by 3 points.
- **Agent close to dead alive**: if an agent was seen close to the killed one while it was still alive (at a close time), then it's suspicious (as it may have killed it later) and its suspect level increases by 2 points.
- **Agent distant from location**: if an agent was seen distant from the location where it said it was (at a close time), then it's suspicious (as it was not seen close to its location at a close time) and its suspect level increases by 1 point.
- **Agent truthful**: if an agent was seen close to the location where it says it was (at a close time), then it's truthful (as it's telling the truth about that) and its suspect level decreases by 1 point.
- **Agents close**: if two agents were (seen) close to each other (at a close time), then they are not suspicious (as they were together) and their suspect level decreases by 2 points.

3.2.2 Queries

All the queries implement the interface `IRBSQuery` provided by the RBS code snippets from the slides of the course. They each implement a method Predicate that returns the predicate to be applied to the provided list of data (the database or data already filtered by previous queries).

The queries implemented for the RBS conditions are:

- **Binding queries**:
  - `DeadBody`
  - `DeadAlive`
  - `LocationInfo`
  - `AgentInfo`
- **Condition queries**:
  - `LocationTime`
  - `SeenLocationTime`
  - `DistantLocationTime`
  - `CloseAgents`
- **Repetition queries**:
  - `CloseDeadBody`
  - `CloseDeadAlive`
  - `Distant`
  - `Truthful`

Table 3.1: Rules of the RBS

Rule	IF **(binding)**	AND **(condition)**	AND **(repetition)**	THEN **(new info)**	AND **(suspect level)**
Agent close to dead body	dead body in ?location at ?time	?agent in close ?location at close ?time	NOT ?agent close to dead body in close ?location at close ?time	?agent close to dead body in close ?location at close ?time	?agent suspect + 3
Agent close to dead alive	* saw dead alive in ?location at ?time	?agent in close ?location at close ?time	NOT ?agent close to dead alive in close ?location at close ?time	?agent close to dead alive in close ?location at close ?time	?agent suspect + 2
Agent distant from location	?agent in ?location1 at ?time	* saw ?agent in ?location2 at close ?time AND ?location2 distant from ?location1	NOT ?agent in distant ?location2 at close ?time	?agent in distant ?location2 at close ?time	?agent suspect + 1
Agent truthful	?agent in ?location at ?time	* saw ?agent in close ?location at close ?time	NOT ?agent truthful in close ?location at close ?time	?agent truthful in ?location at ?time	?agent suspect - 1
Agents close	?agent1 in ?location at ?time	?agent2 in close ?location at close ?time	NOT ?agent2 close to ?agent1 in close ?location at close ?time	?agent2 close to ?agent1 in close ?location at close ?time	?agent1 suspect - 2 AND ?agent2 suspect - 2

3.3 Voting

Once the maximum number of turns is reached, the voting session begins. Each agent votes the agent with the maximum number of suspect points (except for itself). If more than one agent share the same maximum suspect level, then the one to vote is chosen randomly between them.

3.4 Outcome

Once each agent shared its vote, the agent with the maximum number of votes is expelled out of the group. However, if more than one agent have received the same maximum number of votes, then there is an ex aequo between them, and so no one is expelled.

If an agent is voted out and it is the killer, then the other agents win the game. Otherwise, if no one is voted out, or if the voted out agent is not the killer, then the killer wins.

Chapter 4

Usage

4.1 Scene

In the editor `Scene` tab, you can set various parameters in the scene objects and the prefabs.

While the application is running, you can see the `Gizmos` about the agents, such as their name, FOV and LOS, the names of locations and POIs and the current agent that has to accomplish a task in a specific POI.

4.2 Play

In the editor `Play` tab, during the roaming phase, you can move the camera with `WASD` and the directional keys and zoom in and out with the mouse scroll wheel.

During the meeting phase, you can press the `Return` key to go on to the next turn.

4.3 Build

In the build, you can do the same as in the editor `Play` tab.

Additionally, you can press the `Escape` key to quit the application at any time.

Chapter 5

Results

The results show that the agents cannot always identify the killer, but this is the wanted outcome, as they have to simulate a human being behaviour, and so they have to be fallible and reason like humans, who have not all the information required to know the truth.

As expected, the more agents are present in the simulation, the more likely they will fail in identifying the killer, as human players would do. This applies also to high number of turns in the meeting and high kill cooldown, as they will leave the agents acquire much more information not relative to the killer and away in time from the kill.

Chapter 6

Possible Improvements

The project can be improved in almost any of its aspects, in order to make it more plasubile or more efficient, or to add a brand new feature from scratch.

These are the aspects that were found more likely to be improved during development, but left aside as they were not the main aim of the project or they would have made it too complex for its scope:

- Different maps, in order to test the agents' behaviour in a different context.
- More efficient order of tasks.
- Intelligent behaviour during the roaming phase, with agents investigating suspects and the killer avoiding to be seen.
- More turns, so more meetings and the need to store past information.
- A Finite-State Machine (FSM) and/or a Behaviour Tree (BT) to manage the various phases of the meeting.
- More rules and queries in the RBS (and so more information retrieved and stored).
- Personal suspects for the agents, which can reason on their own and weight the information shared by others, based on their level of suspicion.
- Adding player(s) controlled by a human.

Chapter 7

Conclusions

This project shows that implementing an AI agent for social deduction mechanics can be very difficult and that even tiny details are important to make them believable: the key is to find a solution that makes them not all-knowing, but also not too blind with reference to the other agents and their interactions. They have to be concious of the world around them, but believable in their reasoning, to make them appreciable by a human player that may want to interact with them.