Capture the flag: Multi-agent system simulation

Developed by:

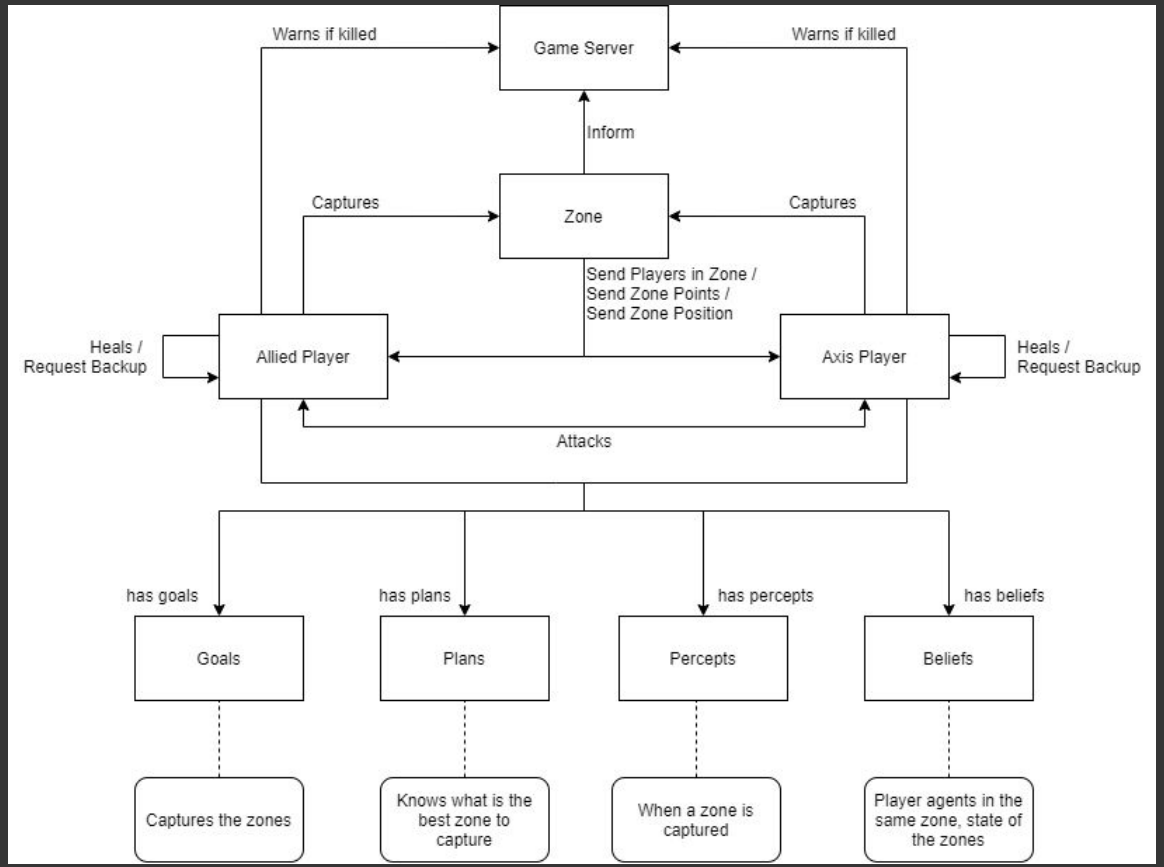
- José Guerra up201706421
- Luis Ramos up201704243
- Martim Silva up201705205

1. Intro

Cooperative, multiplayer action gamemode, revolving around large-scale territory battles over map control zones, inspired by Battlefield and Deep Mind similar modes.

- ➔ **At the start, both teams ...**
  Have the same number of tickets, which denotes the number of times a team's player can respawn.
- ➔ **To win, one of the teams must ...**
  Reduce the enemy's tickets to zero or have more tickets than the enemy by the end of the round.
- ➔ **To reduce the enemy's ticket you need to capture the zones!**

Agents Schema & Communication



Interactions and protocols

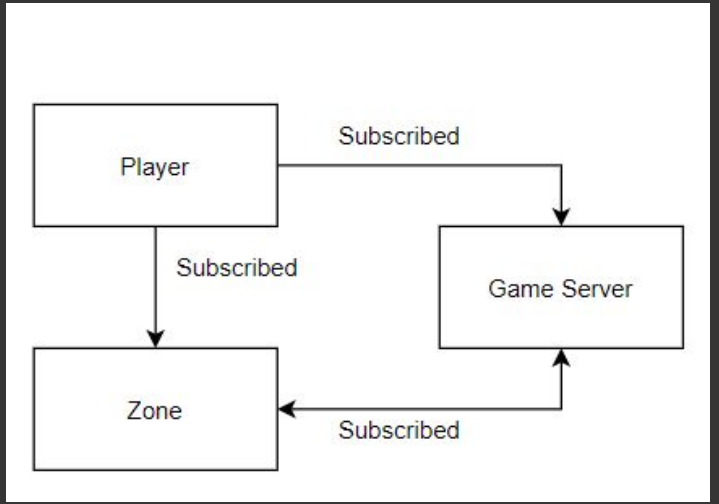
- Players communicate between themselves using **ACL Inform messages**, to inform team members of their current health, inform enemies of attacks, inform allies of heal actions. Also, they use **ACL Request messages** to request for backup.
- Players communicate with zones, via **Inform ACL messages**, to get their position (at the start of the game) or to get the players in the player's current zone.
- Since all players are **subscribed** to all zones, all players get "public" information, regarding the zone's current state.

Usage of FIPA Subscription Protocol

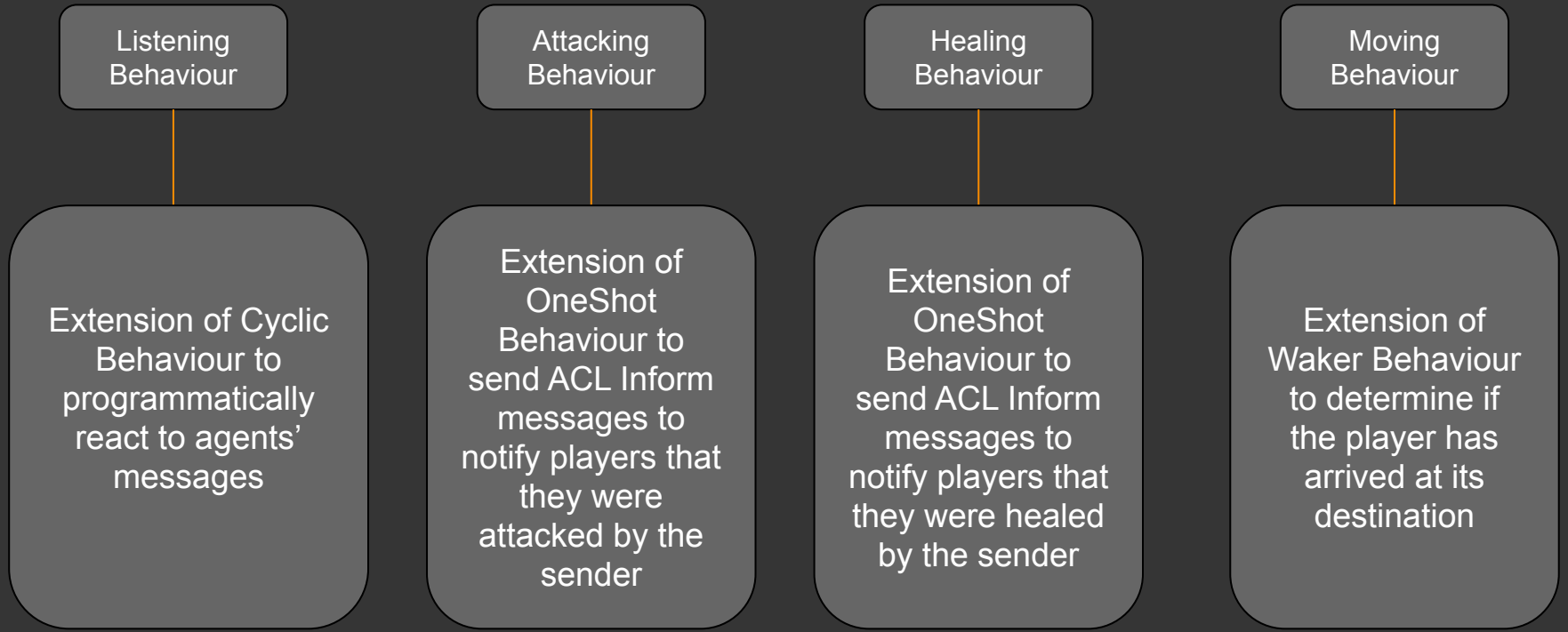
Usage of ACL Inform and Request messages

Interactions and protocols

- The game server is also **subscribed** to all zones, to get their current zone points and determine either or not the game has ended.
- Each player and zone is **subscribed** to the game server, in order to receive start and end of game messages.
- Each player sends an **ACL Inform message**, in case of being killed, to notify the game server in which case the team of the player is deducted 1 ticket.



Player Behaviours



Player Behaviours (cont)

Zone Subscriber	GameServer Subscriber	InformHealth Behaviour	InZone Behaviour
Extension of SubscriptionInitiator to handle subscription to a zone.	Extension of SubscriptionInitiator to handle the subscription to the game server	Extension of OneShotBehaviour to send a message to the team players in the player's current zone with its health current value	Extension of TickerBehaviour to determine the best course of action for a player based on the zone's state and the players in that zone

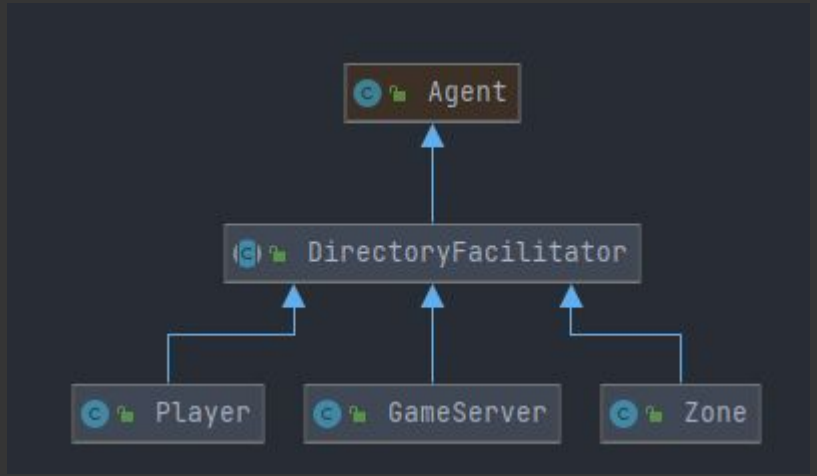
Directory Facilitator

The Directory Facilitator agent class provides 3 simple yet crucial methods that allow every other agent to register and search the DF (yellow page service) for other service providing agents. All agents have access to these methods due it being an abstract class that extends the Agent class.

All agents register in the DF:

- GameServer - game-server
- Player - axis-player or allied-player
- Zone - axis-spawn, allied-spawn or zone

These services are used mainly in the beginning of the game to make sure all agents are ready for the game to start, since an agent is only ready to start when he is registered in the DF service.



Experiments & Result Analysis

We have done through tests and execute different games (can be seen on the execution examples slides) having reach the following conclusions:

- ➔ The Medic class is OP (over powered) on a individual level since it can both attack and defend. This contributes to a higher player individual score.
- ➔ Since all players use the same function to determine the best zone based on its utility, what can happen sometimes, is that, all players from the same team can make the same decision and go all together to the same zone.
- ➔ Since the MovingBehaviour is done using a WakerBehaviour this makes the agent unaware of the changing zones situations while its moving.
- ➔ The Defender class is very weak since its main objective is to stay and defend a zone. This can lead to the partial stop of the agent participation on the overall game.
- ➔ Since we introduce some randomness, to make our game as immersive as possible, we had to test for each strategy multiple times to make sure we extrapolated the correct data information.

Conclusion

We have successfully implemented a version of the popular Battlefield gamemode (conquest) using a MAS.

➔ **Objectives**

The most important were achieved!

➔ **Milestones**

Using agile methodologies we were able to split the work evenly between the developers and do incremental work everyday.

➔ **What's next?**

We would like to focus on data mining and improve our strategies/utility functions using Artificial Intelligence. Besides that, we intent on improving the flow of the game and introduce more player's classes and behaviours.

Implemented Classes: Game Server

The Game Server class is responsible for the maintenance of the game ticket count and the game time limit. The Game Server agent has 4 behaviours:

- GameServerStart (SimpleBehaviour) - Responsible for ensuring all other agents are ready for the game to start;
- GameServerRun (TickerBehaviour) - Responsible for updating the teams' tickets and stopping the game once the time reaches maximum allowed or when a team's tickets go to 0 or below.
- GameServerReceiver (SubscriptionInitiator) - Handles the subscription to the zones, receives zone change updates;
- SubscriptionResponder - Handles the subscriptions to GameServer agent, used for sending game start and over messages.

Implemented Classes: Zone

The Zone class agent represents a zone in the map. This zone can be of 2 different types:

- Base - serves as a spawn point for each team, only 1 spawn point per team;
- Capturable - serves as an objective for Player agents to capture.

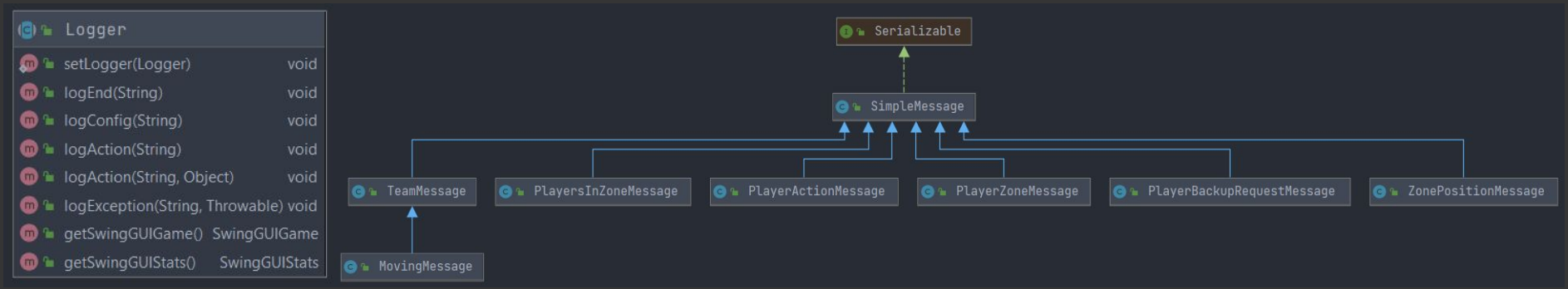
The Zone agent has 5 behaviours:

- GameServerSubscriber (SubscriptionInitiator) - Handles the subscription to the Game Server agent, receives game start and game over messages;
- CapturingBehaviour (TickerBehaviour) - Responsible for updating the zone's status, updates the zone capturing status according to players in the zone.
- InformPlayersInZoneBehaviour (OneShotBehaviour) - Informs all agents in the zone about the existence of one another. Is triggered when the agents in the zone change;
- InformPositionBehaviour (OneShotBehaviour)- Informs all player agents in the game of this zone's position. This behaviour is only used in the beginning of the game.
- ListeningBehaviour (CyclicBehaviour) - Handles the reception of MOVING messages, whether they are left or entered messages by the player agents.

Implemented Classes: Others

For this project other classes were needed to implement in order to have a good sense of what is happening in the game and also for communication between agents. Among these classes are:

- Logger - Responsible for providing the agents with methods that allow for a quick and easy understanding of the game state through the GUI and a log file.
- SimpleMessage - All messages extend the SimpleMessage class which comprises of the message type and the agent type (zone, server, agent).
- *Message - Any other messages serve a specific purpose. These messages are set as the content of the JADE ACLMessage class either to send as an Inform or Request Message



Instructions

We have created scripts for both Linux and Windows Powershell terminals. There are scripts to cleanup, compile and start the game. To make make the game as entertaining as possible you can configure the game settings to experience different outcomes.

➔ Cleanup & Compile

>> scripts/cleanup | >> scripts/compile

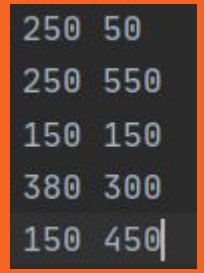
➔ Start

>> scripts/start [zones] [allied] [axis] [initial tickets] [max game time] [num of games]

Example: >> scripts/start 1.txt 1.txt 2.txt 100 200 1

➔ Text files

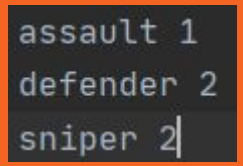
The arguments marked as [zones], [allied] and [axis] refer to text files. This can be changed to experience different outcomes. To change the number of zones and their positions add a file in /zones/ and pass the filename as argument. The same can be done to each team composition located in /players/ where you can edit the number of players for each team and their respective classes (assault, medic, sniper and defender)



/zones/1.txt

Each line corresponds to a different zone. First 2 zones are allied and axis spawn locations.

The two values in each line correspond to the position of the zone (the x and y values respectively)



/players/1.txt

Each line corresponds to the number of players of a certain player class.

The file gives a team's entire composition

Execution Example 1

To run this example:

>>scripts/start 2.txt 3.txt 4.txt 100 250 1

In this example, the allied team has 3 medic, 1 assault, 1 defender and the axis team had 3 sniper, 1 assault, 1 medic.

The axis team ended up winning 100% of the games against the allied. With this we can conclude some things with some degree of certainty.

The axis team compositions is proved to be superior, when stacked against the composition of the allied team. This can be explained, by the fact that there are more offensive units in the axis team. Even though the medic is a unit which can attack and defend at the same time, there are some penalties associated with this class, such as, less velocity in travel time and less damage perform on certain attacks. The defender is proved also to be a less effective unit because its job is to protect conquered zones, which can lead to contribute less to the overall game..

With this, we can analyze that having to many defensive units is not recommended.

```
medic 3
assault 1
defender 1
```

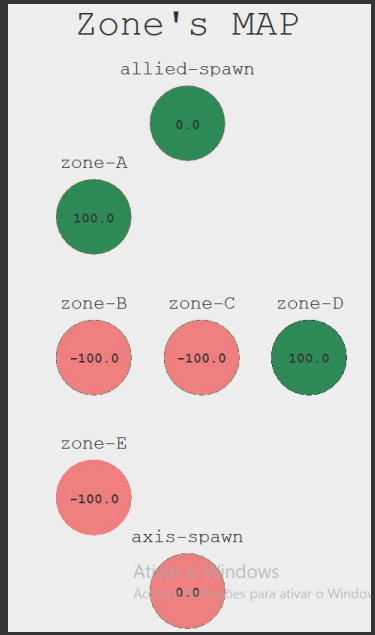
Allied Team Composition

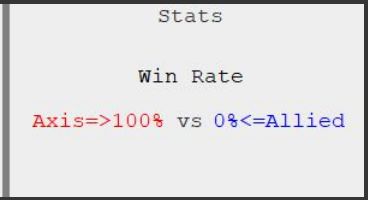
```
sniper 3
assault 1
medic 1
```

Axis Team Composition

```
250 50
250 550
150 150
150 300
265 300
380 300
150 450
```

Zones position





Win Ratio

Execution Example 2

To run this example:

>>scripts/start 2.txt 1.txt 2.txt 100 250 1

In this example, the allied team has 3 assault, 2 sniper and the axis team has 2 sniper, 1 assault, 1 medic.

The allied team ended up winning 100% of the games against the axis. With this we can conclude some things with some degree of certainty.

The allied team compositions is proved to be superior, when stacked against the composition of the allied team. This can be explained, by the fact that there are more solder units in the allied team.

Even though the axis team is more balanced in term of defensive and offensive units, the low number of soldier unit is proved to be an handicap. Since the soldier unit has an advantage in terms of velocity, it can reach any zone quicker than any other unit and this proves to be a great advantage.

```
assault 3
sniper 2
```

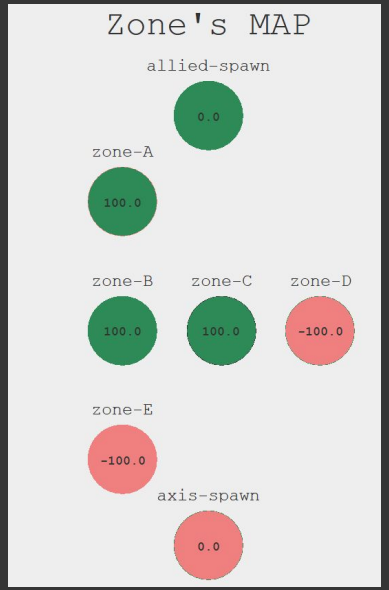
Allied Team Composition

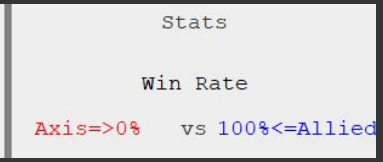
```
assault 1
sniper 2
medic 2
```

Axis Team Composition

```
250 50
250 550
150 150
150 300
265 300
380 300
150 450
```

Zones position





Win Ratio

GUI

The GUI helps to better understand what is happening in the current game and informs about previous games results and stats.

In one frame it is displayed the current game information with:

- Game number
- Game time
- Tickets remaining for each team
- A list of logs related to players interactions
- A zone's map which is a visual representation of the state and position of zones
- A button to switch between zone information and team composition tabs.

GUI (cont)

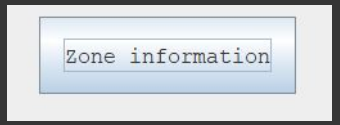
Still on the same frame, to the left there is a section dedicated either to zone information about where the player are or the teams composition. To switch between these two tabs there exists a button that is either labeled “Zone information” or “Team Comp”.

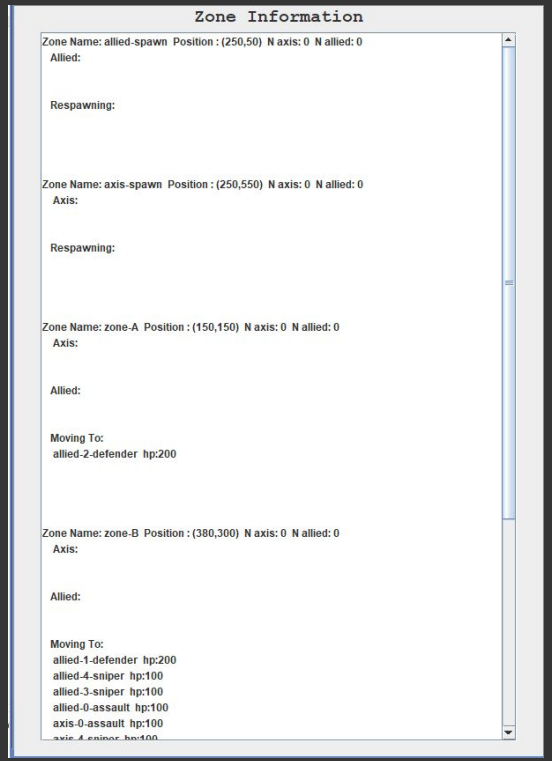
In the team composition tab, the following information is displayed for both team’s players:

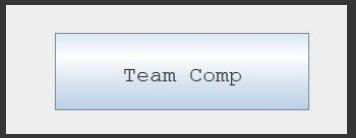
- Name
- Class
- Current Points

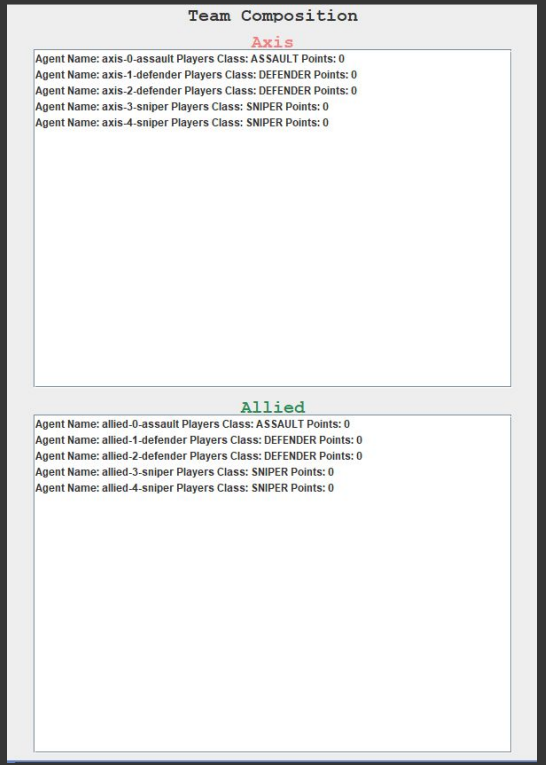
In the Zone information tab, there are sections for each zone with the :

- Name of the zone
- Position of the zone
- Number of players of each team in the zone
- Specific players in that zone or coming to that zone and their health points









GUI (cont)

In another frame it is displayed information about previous games and stats about them such as win rate percentage for both teams. The information about previous games boils down to:

- Game number
- Allied Tickets remaining
- Axis Tickets remaining
- Team that won the game