Smart Applications 2020/21

Software Architecture

Team members: Elia Bisconti, Diletta Goglia, Matteo Montagnani

January 15, 2021

Introduction

In this document we will describe our final development artifact, that is our agent. We will refer to our agent in this document as Karen.

We will dwell on describing our main decisions, motivating our choices and explaining our process of reasoning in detail. In doing that, we refer to the final version of the product (Beta) as the whole process has undergone constant and continuous changes and improvements over time.

We will insist on Karen's strengths, and we will analyze any weaknesses in the final "future improvements" section.

We have chosen Python as the main language for the development of the whole product, as it is what we all feel most confident about. Furthermore this choice did not limit us because we had easy access to everything we needed (manuals, documentation, packages, …) in order to do the best possible work.

We structured our work on different abstraction levels: we started by reasoning at low-level, with entirely rule based model that performed simple movements (in the first try we had random movements) and we added step by step more complex features and components, like cellular automata, fuzzy rules and some simple machine learning modules. This structure can be found in the architectural graph in the next session.

1. UML diagram

In this section we will show the final architecture of our component, through an UML diagram.
Instead of giving a plain description of this diagram, we think it is more interesting to propose here a comparison between the first architecture presented at the beginning of the course, and the new diagram with the final architecture of Beta version.

In this way we can analyze the development of the architecture over time to make a posteriori considerations on how it has evolved.

The first thing that can be seen from this comparison is the absence of the interface between our agent and the League Manager component. This comes from the fact that the communication between these two modules is also available via chat, in a much easier way, without the need of having a dedicated component.
Another important aspect is that the smart part in the movement component is represented by the FuzzyLogic based movement, due to the fact that implementing a neural network capable of moving our agent coherently may be too hard for time reasons and for the quality of the data available.

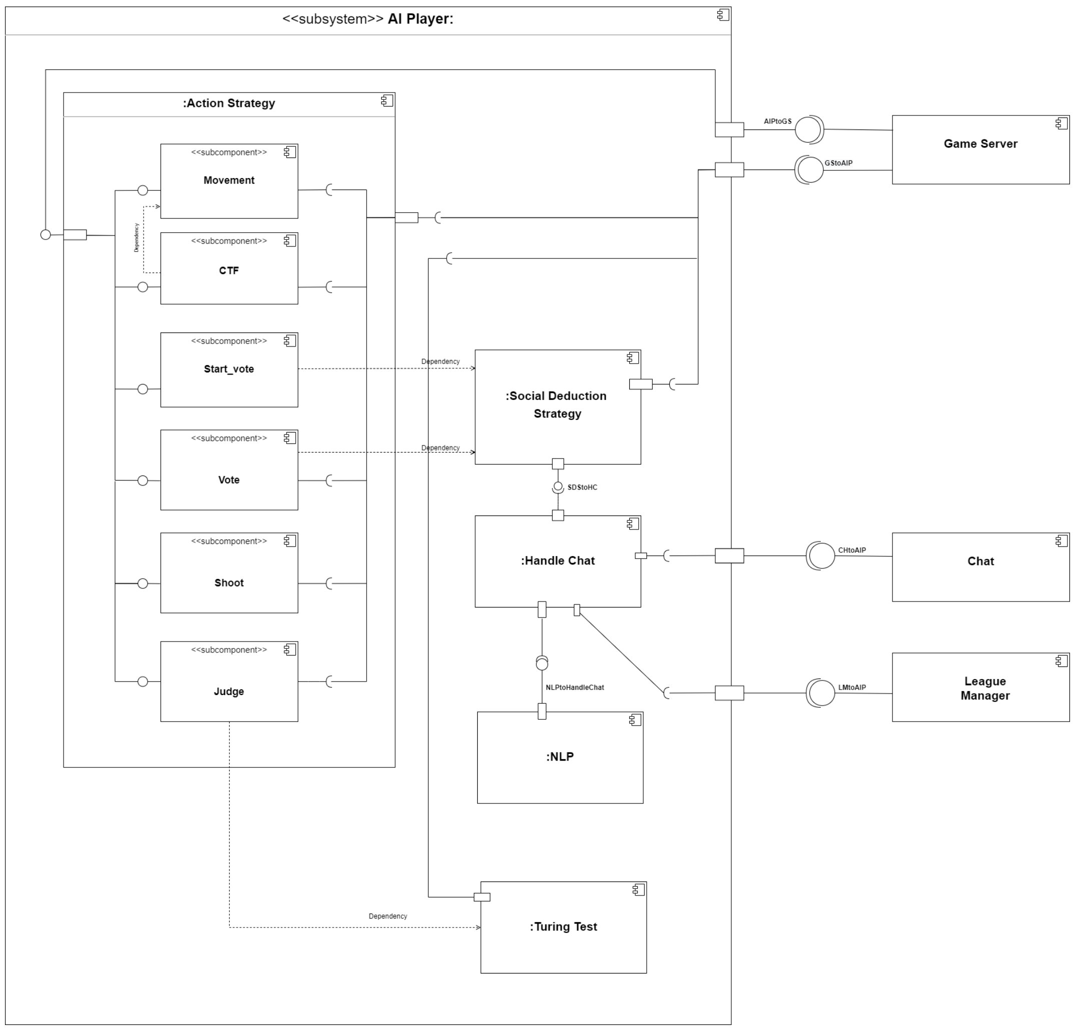

Fig 1. Final Model Architecture

2. Description of the theory of operation

We imagined our Karen as a predominantly **defensive player**: since the results of the first SmartCup already proved this choice as successful, we decided not to abandon it, rather to pursue it.

We therefore modeled our Karen in such a way as to protect herself from shooting, move and kill while avoiding as much as possible the risk of exposing herself to danger, with the ultimate goal of **staying alive as much as possible**.

A hypothetical human user who plays against Karen will therefore not be faced with an attacking player from which to defend him, rather a player to attack because she is able to greatly defend herself.

Assuming that the actual behavior on the battlefield is determined in a fuzzy-way (so that the outcome is not exactly predictable), we predict that Karen's behavior is to move cautiously as described above.
Now let's go into the details of the explanation of Karen's behaviors and reactions in the game. The general tendency is to approach the flag cautiously.
In case she is near an enemy line of fire, she will try to get to safety by moving.
In case she is unable to avoid the danger, and therefore she inevitably ends up under attack, she will shoot in the correct direction without further reasoning or computations. Only in favorable and protected situations will she try to shoot deliberately and not because she is in danger.

As we will explain later, the actual action chosen and performed is in direct control of a fuzzy system and therefore there is a strong component of unpredictability, which strictly depends on the space-time context in which Karen is. Precisely, the fuzzy system decides which is the next action to perform and the outcome action is then actually executed by a low-level component .

As regards the social deduction, Karen uses a system based on score. She will look at every kill made by her teammates: if the killed player was another teammate, she will assign a score to that player. If the score of a teammate is greater than a threshold, she will start a vote; if instead the vote is called by someone else, she will vote the teammate with the highest score. Another feature used to augment the score value of a player is related to its position. An impostor is more likely to stay not so close to the enemy flag, due to the fact that he cannot capture it.

As regards the Turing test, Karen will judge other players in a very cunning way: she will look at the moves made by other players and she will count the number of operations made in 1 second. Due to the different time that humans and AIs have to wait between two operations, this would almost surely allow Karen to guess correctly the nature of the other players. We tried to implement more interesting strategies to discern the behavior of an AI from a Human. Initially by observing the movement of a player, we checked whether the path on the map

performed by a player could correspond to a path obtained through the A * algorithm. This approach has been discarded since this behavioral feature was introduced by the PC and WEB groups.
Another technique used to classify a player as a Human, is derived from a very interesting User Story, initially agreed by all the groups. "An AI should not offend and / or use bad language." Through an NLP module that will be described later, we also use this information to classify a player as human or not. We are actually aware that this feature could not be enough alone, without the support of other information; however we believe it can be an interesting part to insert in this context.

3. Technological choices and implementation details

3.1 Fuzzy logic

We have chosen to use fuzzy logic directly, that is, to establish and execute the game strategy.
The power of this model is that, in various situations, the consequents will all be true, with varying degrees of truth (as you can see in the picture below); since they determine directly Karen's behavior in the game, she will perform exactly the consequent with the highest degrees of truth in that precise moment in time and space. We will describe later in detail the output values that you can see in the picture.

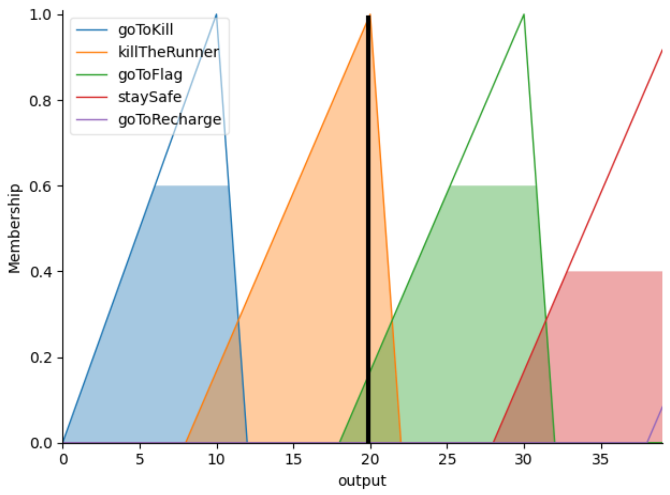

Fig 2. Fuzzy output with different degree of truth

This gave us the possibility of having a **flexible** agent, and so a **smart** agent, in the sense that it **autonomously adapts to all possible configurations** of the game.

In fact, it would be impossible to model these configurations in a rule-based manner, as they are unlimited and unpredictable. Even if it were possible to model a large part of them (perhaps the most interesting or the most predictable), as well as the behaviors that follow, the performance would suffer a lot.

We believe that Karen's unpredictability is a strength: both for her adaptation to the context and for any possible forecasting made by the contenders on her behavior.

3.1.1 Implementation of Fuzzy logic

For the implementation of the fuzzy logic we used the SciKit fuzzy logic toolbox .

Our main Fuzzy Control System (which manages the behavior of the non-impostor player) has ten fuzzy variables, of which 6 inputs (Antecedents) :

- *close_to_enemy*: distance from the nearest enemy's line of fire
- *d_SafeZone*: distance from a safe zone (defined as a cell not in line of fire)
- *nearest_recharge*: distance from the nearest recharger
- *myenergy*: current energy level of Karen
- *d_flag*: distance from the enemy flag
- *runner*: describe the presence of players classified as runners toward the flag
- *stage*: current stage of the game

and 5 outputs (Consequents):

- *goToKill*: Karen will go to kill the nearest enemy
- *goToKillTheRunner*: Karen will go to kill an enemy if he is running to the flag
- *goToFlag*: Karen will move towards the enemy flag
- *goToRecharge*: Karen will move towards the nearest recharger
- *staySafe*: Karen will remain / move towards a safe zone (defined as previously said)

Regarding the “runner” input value, the details will be better explained in section 3.3.

In case Karen is the **impostor**, we have a parallel Fuzzy Control System, in which:

- the outputs are the same as before, except for *goToFlag* that is removed (as the impostor cannot capture the enemy flag)
- the input are the same as before, except for *d_flag* that is removed and except for these two added:

    - *close_to_ally*: distance from the nearest ally's line of fire. It will be useful for when the impostor will start killing allies.
    - *alive_allies:* the ratio between allies that are still alive and the total number of allies at the beginning of the match. Our impostor strategy, in fact, provides that when this ratio gets below 50%, Karen will start killing her allies. This is thought in order to reduce the chance of being seen and discovered (and accused!).

The input values for the fuzzy rules are not recomputed during the game but simply updated by a separate thread. This helps to avoid Karen to be stuck on “computing” values, ensuring quick actions and reactions.

The phase of defining the fuzzy rules was long and detailed, and has undergone many additions, changes and revisions, in order to improve Karen's game performance more and more, and adapt to all the news during these months (different map sizes, different stages of the game, ...).
However, this does not pose any complexity problem since, despite the lengthy ruleset, the fuzzy control system will execute in milliseconds.
It is interesting to note how, with the fuzzy logic, we were able to model a very complex behavior such as playing a match in this game.

The power of fuzzy systems is allowing complicated,
intuitive behavior based on a sparse system
of rules with minimal overhead.

(Scikit-fuzzy API documentation)

3.2 Cellular automata

We exploit Cellular Automata technology for implicitly handling the spatial component of our "complex world", i.e. our map. In particular, we used it for danger avoidance and seeking coverage from enemy attacks.

First of all we used CA to convert the map provided by the server into a **weighted map**, i.e. a map containing different values for each type of cell (grass, river, trap, wall, ...), in order to induce a precise behavior in Karen about the way she moves. Moving by pathfinder, in fact, the CA will make Karen avoid the traps autonomously, prefer crossing through the grass instead the river, and so on.

Then we used CA also to **induce a "stay safe" behavior**, that is, not to end up in enemy lines of fire, assigning highest values to the cells corresponding to these lines. In this way, as she moves, Karen will be discouraged from walking through these cells.

The excellent functioning of this method led to one of Karen's major strengths, which is **surviving**. As it can be seen from the matches data, Karen remained alive almost every time until the end of the matches.

It is clear that this way of inducing a precise behavior can also have consequences on other related behaviors. In this sense, observing the results of the cellular automata implementation was very interesting, as we noticed very powerful implications.

In fact, just by implementing a simple local rule, we observed an **emergent behavior** in Karen, which is that she hides behind walls without her being explicitly instructed to do so.
In fact, since lines of fire are interrupted by walls, “stay safe” can reasonably mean (implicitly) hiding behind a wall. The same goes for the barriers.

We initially tried to model Karen in such a way that she would consider the explicit use of barriers and walls. Given the random nature of the map, this approach was removed in the final version as the behavior was too unreliable in some cases.

This is exactly why we didn't use walls' information in fuzzy logic. Since we got this complex behavior without actually programming it, the fuzzy strategy did not require any more knowledge about how to exploit walls (as distance, opposite position with respect to the enemy, best times and moments to hide, and so on ...).

3.3 RNN (LSTM)

For our ML model we have chosen to build a Recurrent Neural Network, specifically an LSTM. In fact, our goal is to use time series data obtained from log files, to learn the temporal evolution of players' behavior.
For this purpose, a Long Short Term Memory model lends itself very well because it takes into account the whole history of the timeseries, so as not to lose possible correlations between events even distant in time.

We decided to use a more abstract input feature space, so we codified and divided the aforementioned player behaviors into three classes of interest:

- 0: the player shot an enemy
- 1: the player approached the flag
- 2: the player did not approach the flag (i.e. he/she ran from an enemy, he/she went to defend his/her own flag, exc…)

Then we set a first aim, that is predict a player's next move, i.e. use the available data to make a prediction on an individual sequence of actions (belonging to the single player). This means doing forecasting on our three main cases of interest, i.e. whether (or not) the player will go closer to the flag or whether he/she is more likely to shoot.

As previously said, this prediction is inferred from the context, and so from how the game is evolving globally. Synthesizing and abstractly representing game behaviors during a match, through the hidden states of a LSTM, gives us the opportunity to grasp global trends from the environment, to infer useful information from the context and to exploit them.

Then we set a further aim, that is categorize a player in two classes, runner and sniper, according to its most frequent tendencies of behavior.

In order to determine the trend behavior of each player, we used the highest number of actions performed (history of the actions he/she made so far), together with those predicted by the LSTM.

In this way we categorize him/her according to how many times we observe a certain action among those encoded (selecting the action with the highest percentage to be executed / predicted and use that to characterize the player).

We have chosen to use this further information about the players, deriving from the LSTM output, in the Fuzzy Control System, as an additional element to decide how Karen should behave.

This is what is called an adaptive (or neural) fuzzy system, which infers and decides actions/results, on the basis of fuzzy rules, relying on *data* and not rules (i.e. the DIRO, - Data In, Rules Out - process, exactly what happens into a NN).

The more the available data, the more precise the fuzzy rules.

Kosko B., Fuzzy thinking.

3.3.1 Technological implementation of the LSTM

To deploy our network, we used the *Keras* library in the Python environment, in particular the *Sequential()* function.
Starting from this, we put two layers in our network:

- LSTM layer: made of 1 neuron, using Relu as hidden function
- Dense layer: made of 3 neuron (this number will be explained in a while)

As mentioned before, we are interested in getting the next action of a player based on the previous actions that he made. In order to obtain this, we extracted from the log files the sequence of actions made by each player. These sequences have been preprocessed and normalized, in order to have the same length and to reduce the dimensional space. The most interesting parts of these series regard the last stage of the game (or the last part of their life), in which every player should do his/her best.

In order to feed the network with the data described above, we had to encode them in a suitable way. For this reason we decided to use the *one-hot encoding,* obtained through the Keras' function *to_categorical.* This also explains the reason why the dense layer of the network has 3 neurons: one for each bit of this encoding.
Once all of these preliminary operations were done, the size of our available data was 320.960 samples for training and 80.240 samples for validation.

We tried the model on a high number of epochs, but we noticed that the validation loss was not improving much, with moreover a high risk of overfitting.
So we decided to limit the training phase to run for 5 epochs, and here it is a graph that shows the training and validation loss.

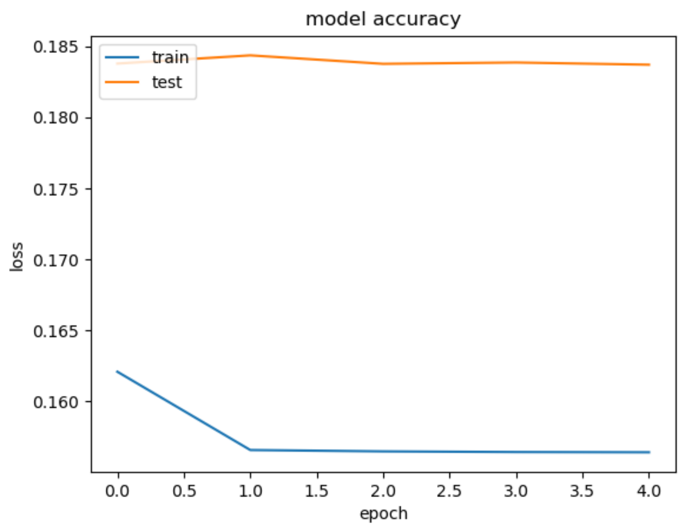

Fig 3. Loss of the model

3.4 NLP module

As mentioned in section, 2 we decided to include a NLP module in order to make Karen able to detect offensive language from other players and tell them to be more friendly. As said before, this also matches our user stories, in which we have the fact that a kid who wants to play AmongAIs should not be reached by offensive language.

3.4.1 Technological implementation of the NLP module

Since Karen is a female player, we focused on realizing a model that predicts misogyny content in the chat.
The dataset used for the training is composed of 5000 labeled tweets that contain (or not) offensive words. This dataset is available at EVALITA, upon request.
Firstly we preprocessed all the tweets to transform them into plain text, removing emojis hashtags and so on.
After that, we tried different feature representation methods: Bert, Tf-idf and Glove. Despite Tf-idf was not the very best choice (precisely the second best one) it is slightly faster than the others (with a minimum loss of accuracy), a key point in a real-time game, for this reason we use this.
We tried logistic regression, random forest and SVM, using a Grid Search to iterate over all the parameters. At the end we found that logistic regression was the best model.

4. User manual

Karen was developed using PyCharm IDE.
To correctly run Karen, the interpreter needs to use Tensorflow v.1.1.15 and Keras 2.3.1. There is a configuration file named *config,* in which there are some fundamental parameters for the connection to the game server and to the chat server. In particular these parameters include the address of the host and the port which connect to.
Running the main, you can decide what to do following the instructions on the screen (i.e. register to a tournament, let karen join a match and so on)

5. Final comments & reflections

The whole work process of these months has strengthened us on many aspects, which we can divide into these macro-areas:

- Cooperation, collaboration and teamwork: we have built a full-collaboration in terms of subdivision of tasks, equable decision-making, work splitting and parallel development of components that must fit together well.

- Quick and consistent adaptation to upcoming changes: adapting quickly and optimally to changes was not easy, but it is a very useful skill that we have learned.

- Competitiveness: work in view of a competition and of a comparison of our work with that of our colleagues, while keeping the "secrets" of our development but at the same time helping and supporting us with other groups.

- Pursuit of new approaches and solutions: we often found ourselves in situations where we knew *what* we had (or wanted) to do, but not *how* to do it. We were therefore forced to leave our comfort zone of knowledge and approach, to seek new solutions and change our point of view in dealing with problems.

6. Further improvements

In general, most of the goals that we set have been achieved. Despite that, we are aware that we can improve many aspects of our agent.

One of these aspects, which unfortunately has taken a back seat in the evolution of the game, concerns the use of Natural Language Processing. We think that this part would be very interesting to develop but we are aware that it might not be very useful for the game due to the fact that the chat is generally little used.

Regarding the improvements of the agent's behavior during the match, the most important thing that we could improve is the agent's interaction with the objects. It was one of the first parts we focused on, but an entirely rule-based technique seemed to be quite unreliable. A possible approach would be the use of a CNN or of an autoencoder to analyze the map composition during time.

Another possible improvement could be the extension of our LSTM model with the use of a Kohonen Network in order to perform a better unsupervised classification/clustering of players.