MiniClashAI report

BOISMARTEL Léo

July 2023

Summary

1 Introduction

1.1 Why starting this game

The idea of creating this game sprouted in my mind when I witnessed the advancements in the field of AI. While studying in Canada, I frequently utilized ChatGPT to gauge the capabilities of AI, and I was genuinely astonished by the sheer power and comprehension exhibited by the machine. Coincidentally, during that time, I also indulged in playing ClashMini, a game developed by SuperCell and I was thinking that it's a game probably easy to predict with AI. I was thinking that because the game is an array of 8x5 cells that are determined at the start and then the game plays on his own. The result is a win or lose for both teams so it can be predicted.

As I delved into the mechanics of ClashMini, I envisioned how my own game could function and eagerly embarked on coding it according to my vision. This project became a personal endeavor, providing me with an opportunity for extensive Python and C++ training exercises. To incorporate the AI aspect, I employed the TensorFlow library in Python, which proved instrumental in implementing the AI functionalities. Additionally, I opted for C++ due to its Object-Oriented Programming paradigm, as it perfectly aligned with the requirements and scope of this project.

Although it is currently an individual project, I acknowledge that there is ample room for improvement and expansion. With further enhancements, this game can reach even greater heights and offer an enriching experience for players.

1.2 How the game work

Now, let's introduce the game and how it works. MiniClash is an automated game where the only interactive part is the placement of troops. Unlike the original game, which consists of three rounds, this version has only one round, making troop placement crucial.

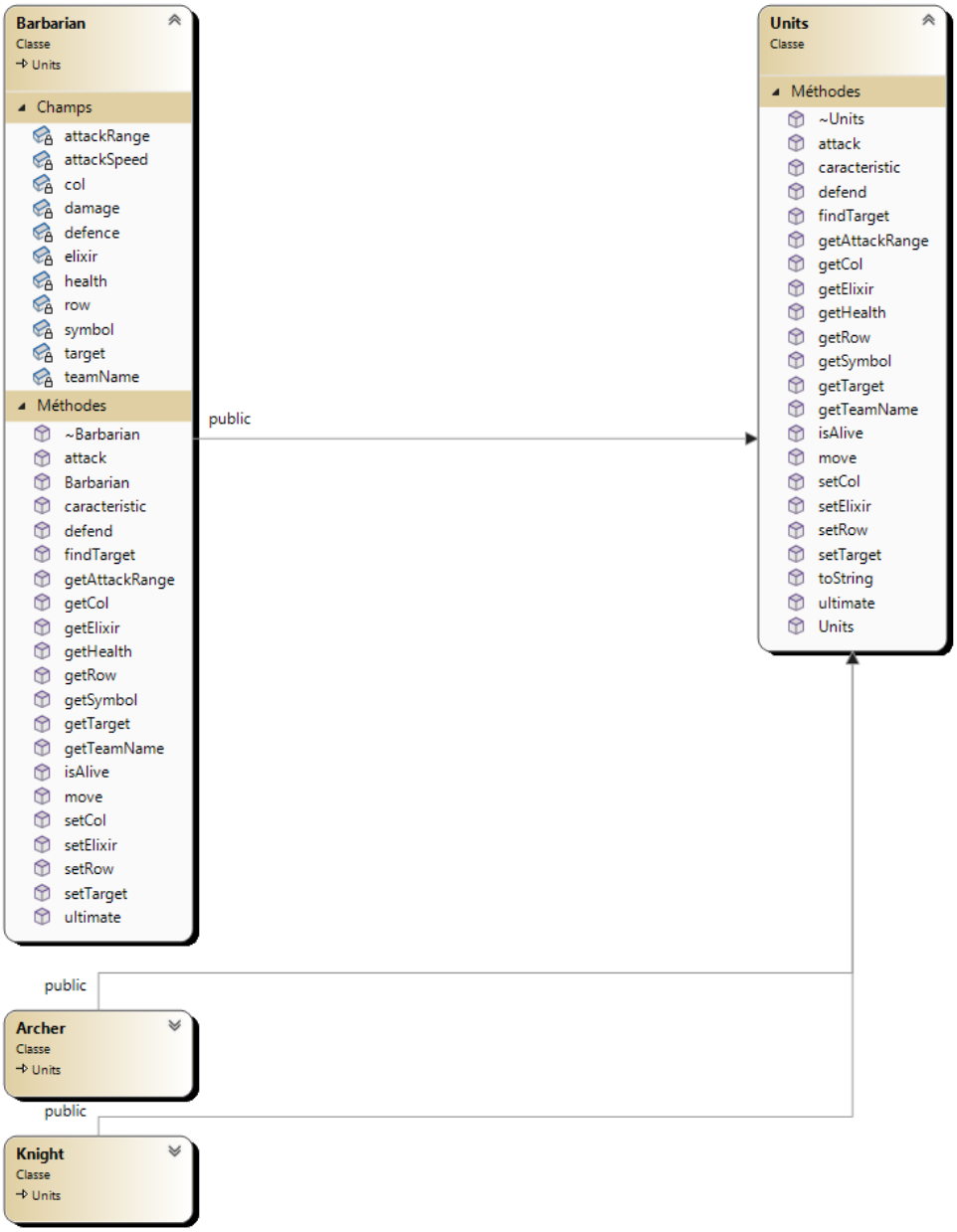
To deploy a troop, players need elixir, which is allocated at the start of the game. Each player starts with 6 elixir, and both players have the same amount. It's important to strategically manage elixir for troop placement.

In this initial beta version, there are only three troops available, each costing 2 elixir. Consequently, all troops can be placed on the battlefield since the total available elixir allows for their deployment.

2 Units development

2.1 Class structure

The first consideration was the implementation method of the units. These troops are represented by the base class 'Unit,' which defines all the possible troop attributes to be overridden by derived units. Next, an arena was created to contain the troops. It is represented by an array of 8 arrays of 5 'Unit' pointers. Since each team can have only 3 troops, many of the pointers are null. The troop class includes attributes such as position in the array, damage potential, health, target, symbol, team name, and elixirs. This class also has variables for defense and attack speed, although these have not been implemented yet. In addition to these attributes, there are various functions that can be called, such as the constructor and destructor of this class, the troop's action, a function to find a target, and other functions that provide information about the troop.



2.2 Class methods

I won't discuss the getter and setter methods at this time.

Currently, all classes have an "attack" function, which deals damage to a target. The "Arena" class calls this function when the target is within the attack range. If the attack results in the target's death, it is removed from the grid.

The "defend" function prioritizes using the troop's defense before its health, but so far, there are no troops that have a defense attribute.

Finding a target is a specialized function. For close combat troops, it searches for the closest troop, while for long-range troops, it searches for the farthest troop. The selected target is then stored as an attribute of the unit.

The "movement" function utilizes a path finder in the "arena" class and executes the first movement only. It swaps the position of the troop with the adjacent empty cell since moving to a cell occupied by another troop isn't possible. The troop's new position is then updated in its attributes.

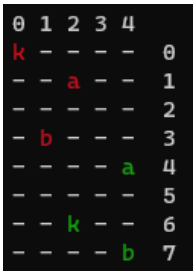
Here's an example of initialisation of the troop.

```
Knight::Knight(int row, int col, char teamName) {
    this->row = row;
    this->col = col;
    damage = 1;
    health = 11;
    defence = 0;
    attackSpeed = 0.5;
    attackRange = 1;
    target = nullptr;
    this->teamName = teamName;
    elixir = 2;
}
```

3 Arena development

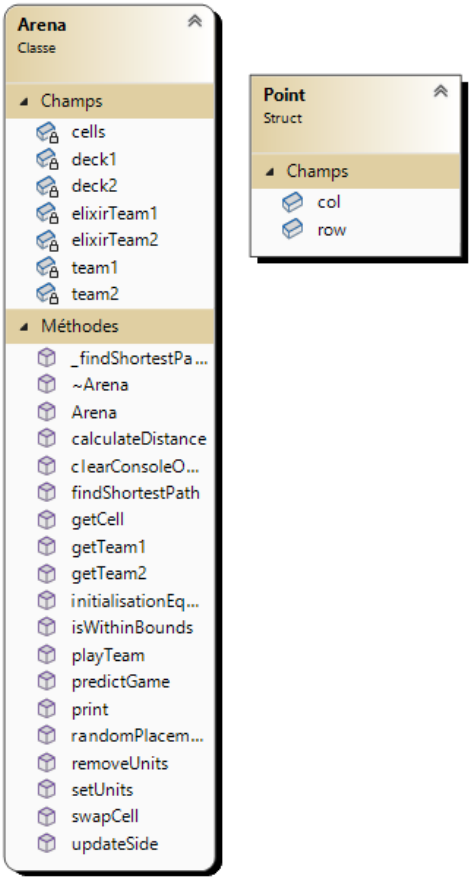
3.1 Class structure

One important class is the "Arena," which serves as the game's core component. It manages various elements such as cells, both team decks, elixir, and the teams already in place.
Regarding the cells, they are represented by an 8x5 array of pointers, forming the game map. If a pointer is "NULL," it is displayed as a dash. Otherwise, it represents a troop and is associated with its own class.



The deck attributes determine which troops can be used at the beginning of the game. In this beta version, there are only three troops available, eliminating the need for choices. Each troop costs two elixir, and once placed, it is automatically added to the team attribute as a list of unit pointers.

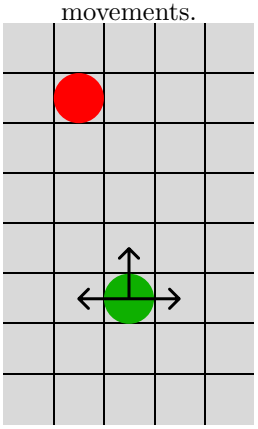
Now the methods of the "Arena" class.



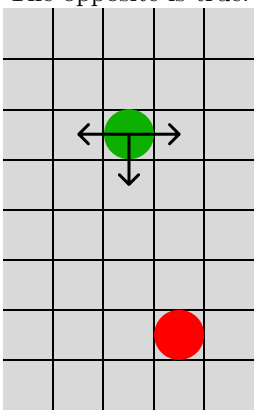
3.2 Class methods

I will skip explaining the basic creation and deletion of the class, as it is straightforward with these attributes. Essentially, the "Point" class within the "Arena" serves to indicate a cell on the board. Now, let's discuss the function that calculates the distance between two coordinates and returns an integer value. Although an integer is less precise than a float, it is sufficient for our purposes. This function is used to determine if two troops are adjacent, within the troop range of attack, so they can engage in combat. To ensure my game functions properly, I needed a function for troop movement. However, I couldn't allow the troop to choose any path but only the shortest one. To address this requirement, I developed a recursive function.

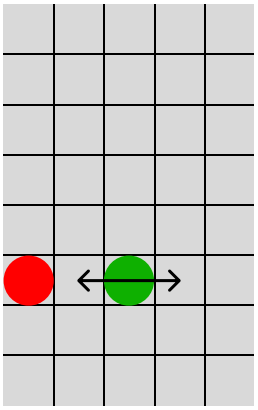
As long as the target position is higher than the current troop position, the troop has three available movements.



The opposite is true.



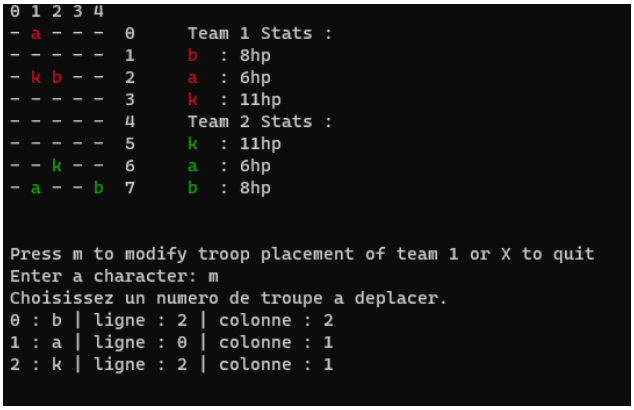
When they are on the same row then there are only two possibilities.



In order to assist this function, I devised a straightforward function that determines whether a given point is within the grid or not.
These methods are crucial for improving efficiency in finding the optimal path. The main challenge I encountered with this function is that although I initially envisioned resolving a maze, my starting and ending points are not enclosed by walls, resulting in multiple possible solutions to reach the end. This creates a complex maze scenario for processing with this approach. Moreover, this method is advantageous because I have other units that act as "walls" and block certain paths.

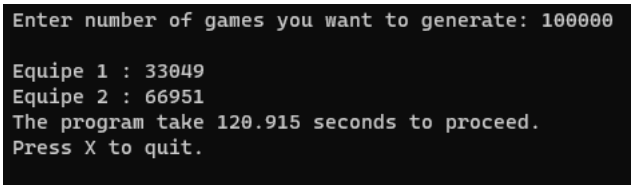
With my actual method another issue I faced is when I attempt to find a path between a start and end point in the same row but with an obstacle in between, which leads to no viable path. To address this problem, I implemented a solution where the unit starts in a cell above or below the obstacle and added that to her path(cell list of movement).

The initial phase of the game is crucial, where proper troop placement is essential for players to adapt easily. To facilitate this, a dedicated function guides the player in making their placements and handles any errors, retaining only a specific number in the well side of the player.



To expedite the game's start, I've created a function that utilizes the maximum elixir of each team to randomly place troops on cells. This function employs C++ libraries "chrono" and "srand." Both teams use this function sequentially. However, an unexpected issue arose when troops on both sides ended up in a mirrored position. This occurred because the chronometer remained the same for both function uses. To address this, I incorporated the high-resolution clock to obtain a more precise seed.

Now, for the game to proceed autonomously, it requires an intelligent decision-making process. This is where the game management function comes into play. It employs a switch statement for each troop on one side to determine the appropriate action at each round. The first possibility is movement, and if no movement occurs, it means the target is within attacking range. The second possibility is to attack. An interesting observation is that team "1" starts playing first, initiating either movement or attack. Even more intriguing is that, over a batch of 100,000 games, team "1" emerged victorious in 33% of the cases.



4 Neural Network

In this section, a Python script is utilized by the main program, operating with the assistance of the "tensorflow" library.

4.1 Data Reading

The initial step involves reading data from a text file. The input comprises a grid, as described earlier, along with a string representing the winning team, either "one" or "two." Upon reading the document, we proceed to split the grid and result into two separate lists. Each game is present in this list, with the grid and its corresponding result located at the same index. However, before using this data, we need to encode each character. For instance, we assign the value 0 to '-' and encode 'a' as 1, 'b' as 2, and 'k' as 3. The win label is also encoded as [1, 0] for 'one' and [0, 1] for 'two'.

4.2 Learning

1. Model Definition: The function "creer_modele()" creates a neural network model using the Sequential API from Keras. The Sequential API allows you to create a linear stack of layers, where data flows sequentially from the input layer through each subsequent layer until it reaches the output layer.

2. Input Layer: The first layer in the model is a Flatten layer. This layer is responsible for converting the 2-dimensional input data of shape (8, 5) into a 1-dimensional array. The Flatten layer essentially reshapes the input data to be suitable for feeding it into the subsequent dense layers. The input data is assumed to have a shape of (8, 5), which means there are 8 rows and 5 columns.

3. Hidden Layer: The second layer in the model is a fully connected Dense layer with 32 units. Each unit in this layer is a neuron that takes input from all the 1D elements (flattened input) and applies a rectified linear activation function (ReLU) to the weighted sum of the inputs. ReLU is commonly used in hidden layers to introduce non-linearity and help the model learn complex patterns.

4. Output Layer: The third and final layer is another Dense layer with 2 units. This is the output layer responsible for making predictions. The activation function used here is softmax, which is appropriate for multi-class classification tasks. It computes the probabilities for each class, ensuring that the sum of the probabilities for all classes is equal to 1.

5. Model Compilation: After defining the architecture, the model is compiled using the compile() method. During compilation, you specify the optimizer, loss function, and evaluation metrics.

    - Optimizer: In this case, the chosen optimizer is 'adam'. Adam is a popular optimization algorithm that adapts the learning rate during training and performs well in various scenarios.
    - Loss Function: The model uses 'categorical_crossentropy' as the loss function. This loss function is suitable for multi-class classification problems and measures the difference between the predicted class probabilities and the true class labels.
    - Metrics: The model is set to evaluate its performance using 'accuracy', which is a common metric for classification tasks. It measures the proportion of correctly classified samples.
    - Model Return: Finally, the function returns the compiled model, which can be used for training and making predictions.

To summarize, this model is designed for a multi-class classification task with 8x5 input data. It consists of a single hidden layer with 32 neurons and an output layer with 2 neurons, using the ReLU activation in the hidden layer and softmax activation in the output layer. It is compiled with the 'adam' optimizer, 'categorical_crossentropy' loss function, and 'accuracy' metric for evaluation.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 40)                0

 dense (Dense)               (None, 32)                1312

 dense_1 (Dense)             (None, 2)                 66

=================================================================
Total params: 1378 (5.38 KB)
Trainable params: 1378 (5.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

After generating a model using the user-generated file, we conduct a training process with 10 epochs. Following that, we evaluate the performance of the model using predefined test datasets.

```
Epoch 1/10
313/313 [==============================] - 0s 559us/step - loss: 0.6639 - accuracy: 0.6325
Epoch 2/10
313/313 [==============================] - 0s 552us/step - loss: 0.6014 - accuracy: 0.6844
Epoch 3/10
313/313 [==============================] - 0s 554us/step - loss: 0.5648 - accuracy: 0.7258
Epoch 4/10
313/313 [==============================] - 0s 541us/step - loss: 0.5264 - accuracy: 0.7576
Epoch 5/10
313/313 [==============================] - 0s 529us/step - loss: 0.4969 - accuracy: 0.7730
Epoch 6/10
313/313 [==============================] - 0s 540us/step - loss: 0.4768 - accuracy: 0.7826
Epoch 7/10
313/313 [==============================] - 0s 543us/step - loss: 0.4640 - accuracy: 0.7859
Epoch 8/10
313/313 [==============================] - 0s 539us/step - loss: 0.4533 - accuracy: 0.7909
Epoch 9/10
313/313 [==============================] - 0s 552us/step - loss: 0.4466 - accuracy: 0.7937
Epoch 10/10
313/313 [==============================] - 0s 536us/step - loss: 0.4414 - accuracy: 0.7993
Predefined tests:
Batch of size: 100
4/4 [==============================] - 0s 332us/step
The success rate is: 78.0 % on the tests.
Press X to quit.
```

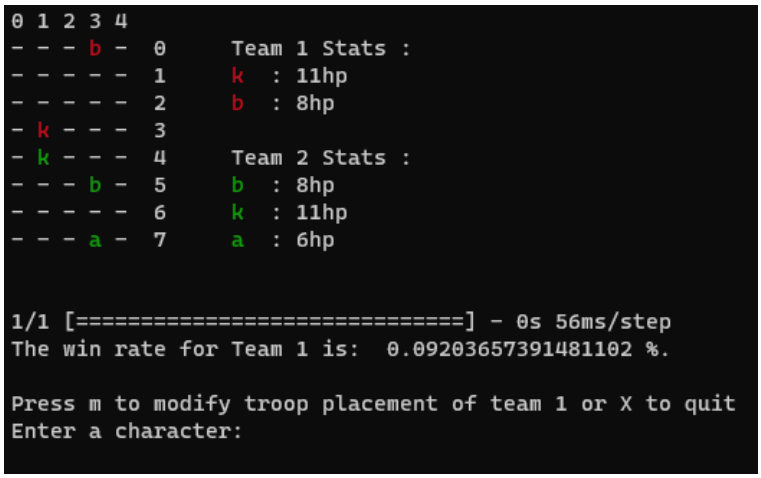
4.3 Exercice

Users have the option to test their model using the generated file. To track model progress, they can generate a smaller file, train the model with it, and then conduct tests using the same file. As long as the user continues with the same model, all the data remains associated with that model. The only means of resetting the model is by creating a new one. Here is an example.

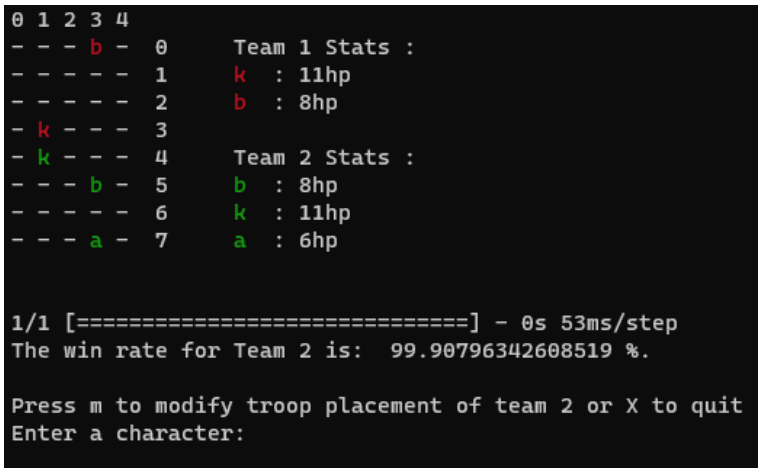
```
Test with 'results' file:
Batch of size: 200
7/7 [==============================] - 0s 882us/step
The success rate is: 94.0 % on the tests.
Press X to quit.
```

4.4 Prediction

The crucial aspect of this program is to utilize the newly created model! Our objective is to predict the winner based solely on the grid. The predictions are made using 'tensorflow' with the provided model. Upon reading the data, the program returns the likelihood of team '1' winning, and we interpret the results to present each team's perspective.



The model is designed to automatically identify impossible-to-win scenarios, where one side has one troop less than the other. These games are not intended to be played, hence why I display the win probabilities. As such, both teams are encouraged to adjust their compositions. If we keep everything unchanged, team 2's win probability should be 100 minus team 1's probability.



The sum of the probabilities for team 1 and team 2 equals 100, as evident from 99.90796342608519+0.09203657391481102.

Let's now examine how a mistake on one side affects the predictions. Here is an example of a poor positioning for team 2.

```
0 1 2 3 4
- - - - -  0     Team 1 Stats :
- - - - -  1     b  : 8hp
- k - - a  2     k  : 11hp
- - - - b  3     a  : 6hp
- - - b -  4     Team 2 Stats :
- - - - k  5     a  : 6hp
- - - - -  6     b  : 8hp
a - - - -  7     k  : 11hp


1/1 [==============================] - 0s 55ms/step
The win rate for Team 1 is:  0.061788043240085244 %.

Press m to modify troop placement of team 1 or X to quit
Enter a character:
```

Next, we show the same team 2 with significantly different positioning.

```
0 1 2 3 4
- - - - b  0     Team 1 Stats :
- - - a -  1     b  : 8hp
- - - - k  2     k  : 11hp
- - - - -  3     a  : 6hp
- - - b -  4     Team 2 Stats :
- - - - k  5     a  : 6hp
- - - - -  6     b  : 8hp
a - - - -  7     k  : 11hp


1/1 [==============================] - 0s 55ms/step
The win rate for Team 1 is:  99.87667798995972 %.

Press m to modify troop placement or X to quit
Enter a character:
```

The underlying tactic is that team 2's archer will target the farthest troop but will perish beforehand due to the team 1 archer, while the team 1 knight takes on the role of a tank.

5 Conclusion

I thoroughly enjoyed working on this project. Tensorflow made it incredibly easy to create an AI, and it was a delightful experience to explore new possibilities. Although I initially had ambitious plans to add more options, I'm content that the project is currently functioning well. It reminds me of my first time using AI in school when I attempted to implement a neural network from scratch; it was quite challenging. However, this time, the process was much smoother and straightforward. With this successful project behind me, I look forward to working on other projects, and AI will undoubtedly be on my mind for future endeavors.