Waventure

LEFEBVRE Romain
Engineering College Leonard De Vinci
Courbevoie, France
romain.lefebvre@edu.devinci.fr

LEDRU Solal
Engineering College Leonard De Vinci
Courbevoie, France
solal.ledru@edu.devinci.fr

HIRTH Quentin
Engineering College Leonard De Vinci
Courbevoie, France
quentin.hirth@edu.devinci.fr

CARRE Arthur
Engineering College Leonard De Vinci
Courbevoie, France
arthur@carreo.fr

***Abstract*—Our project is a software that the user would launch at the start of a battle in the game Waven. The main goal of our software is to be able to automatically complete the battle, by choosing the best actions to take without any human interaction. The software aims at analyzing and understanding the placement of enemies as well as their types and statistics, but it must also be able to read and know the cards available to play. To do so, he will have to perform two actions: move/attack (in Waven the attack is an extension of the movement) as well as cast spells (in the form of cards) to overcome enemies. If we succeed in the project, we can already see multiple ways to improve it. The first is to ensure that the software can launch itself into the game and that it can navigate the menus on its own to choose which fight to carry out. The second would be that he can repeat the same mission a defined number of times to collect rewards in large quantities. We decided to use the easiest character for our software, but it would also be interesting to teach our software to play with characters that have more complicated gameplay. We could also work to implement the more complex features of the game, which require more management of our resources during battle, but it would heavily affect the speed of the AI computing.**

User, Customer	Arthur, Quentin	Download the software from a website and need an easy to use application to help him automating Waven fights.
Software Developer	Solal	Program a software providing all the requirements planned. He's in charge of design solutions to implement the functionalities
Development manager	Romain	Must guide the development team by giving the deadlines, make sure the requirements are well implemented and find solutions for development related issues.

I. INTRODUCTION

A. Motivation

Almost everyone has already played a video game, there are so many genres that everyone can have fun with at least one game. However, some games might require a bigger investment from the players than others, as they ask for a more valuable resource than your money: your time. The act of repeating a certain part of the game is called "farming", and it can be quite an ordeal, depending on if that part is fun or not, but even the funniest part gets boring when repeated. This is why some games implement an automatic way of doing it, where you just spend the resources needed to complete whatever you need to do, and it just gives the rewards automatically, but it is far from a common thing. That is why some programmers decided to take the matter into their own hands and do it themselves by coding programs that will automatically do the actions as if they were the players, but without the need for players. It was while seeing such programs in some of the games we play that we decided to try and do the same for another game we know of: Waven. The program, in both cases, does not involve machine learning as it only finds an optimal list of instructions from a game tree, but it could be added later to polish the fittest function we will be using.

As for our educational background, we were interested in how artificial intelligence can recognize an object (whether it is in a video game or not) as well as how software can take decisions with the best output depending on the situation. This project is an opportunity for us to make a first step in the field of AI and learn a lot about its key concepts.

B. Related Software

When we were looking for other related software, we found AI used on another game from the same studio called Dofus. The main difference here is that the latter does have a player-based economy, so the creation and publication of automated applications add to many resources without effort and ruin a major part of the game. In our case, Waven doesn't give you any advantage over other players regarding equipment or money, so it only allows you to focus more on the interesting

content of the game and skip the boring but time-consuming one.

II. REQUIREMENTS

A. Lauching the software

To launch our application the user must have a computer, which will serve as support to launch the game "Waven" and our software. The user must have previously downloaded python and the libraries needed.

B. Recognize the situation

Once the software launches, it should check if the computer it is running on has another program named Waven running. Then, after checking if the user is in the battle screen, it should be able to extract all the key information from the screen and assign them correctly to the AI variables.

C. Find all possible lists of instructions

Once all the information are obtained, the software must resolve the situation by finding the best list of instructions to make an optimal turn. It will generate all possible board states with the options available for the player. We also need to give a score to the boards to choose the best one and check if the fight is resolved on the board.

D. Choose the right instructions

On one hand, if a board ends the fight, then we want to reach it. On the other hand, if the timer ends without any end-board, we want to reach the most advantageous one.

E. Time limit

A timer may be added to ensure that the software chose a solution before the end of the player turn, avoiding to miss it because of long calculation time.

F. Apply optimal list of instructions

The software must be able to execute the list of instructions previously found using simulated mouse inputs. We can also consider the possibility to run the AI after each action if the running time allows it to adapt against critical hit, moving foes or other random game features to optimize our turn.

G. Recognize new turn

The software must be able to differentiate a turn where it needs to find the best list of instructions and the foes turn where it is just waiting for them to take their actions and move to new positions.

III. DEVELOPMENT ENVIRONMENT

A. Software Development Platform

Our software is entirely built and aims to be used on a personal computer. No purchase of software or hardware is expected at all.

For our development platform, we will be using Windows 11 as it is the operating system we all already have in our laptops and it does provide all the resources needed for our project. The overall performances of the computer must be enough to launch the game, the software won't add significant burden on your hardware.

As for the programming language, it will be Python. The two main reasons are the abundance of resources to help us program, extract the data from the screen and implement the AI, and the overall simplicity of the syntax when it comes to write the code.

As Python is a slow language and we're constrained by a time limit, we might change to C++ if the AI's running speed is too slow. It also has all the resources needed and is significantly faster, but significantly harder to program our software with.

B. Software In Use

In addition of our software, you will need to download the game "Waven" from the official website [1]. Waven is a turn based RPG where you need to resolve combats against your different types of enemies in a small arena to gather equipment, companions and powerful spells in order to face more difficult challenges.

Fig. 1. An image of the board

You will also need character of the right class, level 30 with all the equipment needed. To do so, we suggest you to use our account if you want to easily test the software, please send us an email or directly ask us for ID and password.

C. Task Distribution

Our project will be divided into two main subjects:

First, handle all the interactivity with the game, from extracting the data and store it in the right variable to the implementation of simulated mouse inputs. This task will be completed by Arthur and Solal.

Then, program the AI with all the necessary features in python. This task will be completed by Quentin and Romain.

Finally, write the documentation and manage the GitHub. All the members will help for this according to how they manage their tasks and the relevance of doing it.

IV. SPECIFICATIONS

A. Launching the software

The software will be designed to be launched as an executable file. Once the application is launched, it will give the user a window with two main features.

Firstly, a button linked to a colored chip. Once the button clicked, the software will search for a Waven session running on the computer. Once it is found, the chip appears green, red otherwise or if the button isn't pressed at all.

Secondly, a button to properly launch the AI and find the list of instructions for an optimal turn. Once pressed, the software checks if a battle is engaged and, if so, solve it.

An error message will be shown to the user if he pressed the run button without Waven open,without being in a fight or if the program is already running.

B. Recognize the situation

To gather all the data needed in order to run our AI we must extract it from the game.

We will first scan the 7x7 board to note the free and occupied cells. For all occupied cells, we will create a character object, drag our mouse on it to open a little window with the needed stats and read them. Each time a stat value is read, it's stored in the corresponding variable.

Then we observe the color of a relevant pixel for each spell card in our hand. It allows us to know what spell card is available. Each time a spell card is recognized, its ID is stored in the hand array.

Finally, we drag the mouse on the player portrait to show his stats and extract them as well.

C. Find all possible lists of instructions

Once all the data is extracted, the software will run the AI. It will first set a game tree with our initial board as first node and generate all the final boards we can obtain as new branches.

First we will check all the available cells to move (in blue on the image). Then we will check for all cells connecting to an enemy. The goal here is to perform a melee attack on the enemy adjacent of our character. If at least one enemy is reachable, we won't check the boards with no enemy attacked as it will always be less interesting.

Then we will use all combinations of available spells on the remaining foes to generate all the possible final boards. We have a limit of 6 resources to spend in our spells as we do not implement the bonus resource.

Once all boards are generated, we generate for each of them the enemy turn and restart the process for all boards generated adding another layer to our game tree.

Fig. 2. An image of the spells

Each time a board is generated we launch a fittest function on it to give it a score. A greater score means a more advantageous end board. If the board found solve the fight, its list of instruction is directly applied and the calculus stopped.

Fig. 3. An image of the available cells

D. Choose the right instructions

If no end-board is found after the end of the timer, we need to choose a board from our game tree. We will use the minmax algorithm [2] to find the list of instructions with the best boards overall and be less dependant of randomness on foes turn.

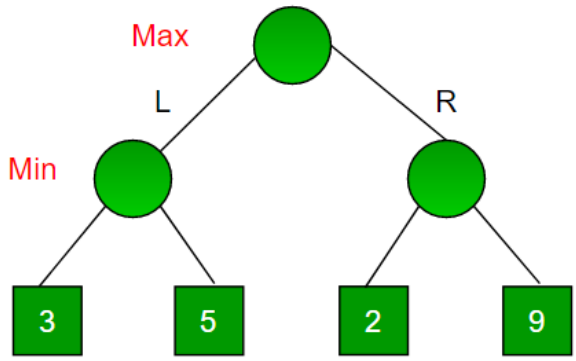

Fig. 4. Diagram of the MiniMax Algorithm

E. Time limit

As soon that the player turn starts, we will launch a timer to chose a list of instructions before the end of the turn and apply it.

The timer duration will be around a 60 seconds as a turn duration is 90 seconds.

F. Apply optimal list of instructions

Once the fit function came to an end, we will apply the optimal list of instructions. The program will go trough each tuple of instruction and launch the corresponding function with the right parameters to directly input them in-game.

Once the list of instructions applied, we pressed the button to end the turn.

G. Recognize new turn

As soon that the program is launched, we will run a while loop checking for clues about who's turn it is. The goal here is to check the color of the pixel forming the player turn's clock to see if it began.

We also check the overall luminosity of pixels on screen. In fact, those ones become darker once the fight resolved. If that's the case, we can end the program and wait for another press on the start button of the software.

Fig. 5. An image of the buttons

REFERENCES

[1] Waven official website - https://www.waven-game.com/fr/season
[2] Geeks-for-geeks, Minimax algorithm - https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/