Zappy Documentation

Release latest

Jun 26, 2022

CONTENTS

CHAPTER

ONE

WELCOME TO OUR ZAPPY DOCUMENTATION!

Zappy is the final project made in our second year in Epitech. The goal of this project is to create a network game. Several teams confront on a tiles map containing resources. The winning team is the one with 6 players who reached maximum elevation. The following documentation describe all the details and constraints.

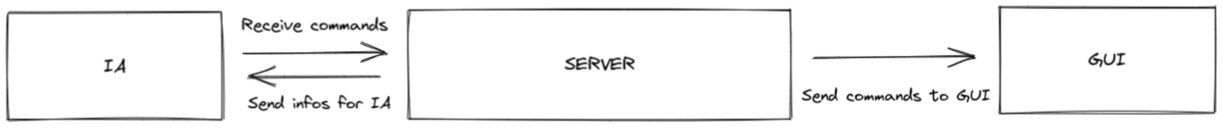
Thus there is three parts to this project:

- **The server:** the project core that manages the entire game in a network.
- **The graphical interface:** interprets the commands received by the server and displays a result on a 3d map.
- **The artificial intelligence:** takes posession of the players to make them level up in an optimized way.

CHAPTER

TWO

SCHEMA



–> Represents the interaction between SERVER/IA and SERVER/GUI.

CHAPTER THREE

INFORMATION

Note: We are a team of 6 students on this project. To divide the work, 2 people are in charge of each of the 3 parts that will be detailed in this documentation.

Warning: This project requires a lot of work during 4 weeks. Time management must be well thought out in order to complete this project on time!

CHAPTER FOUR

CONTENTS

4.1 Usage

To use Zappy Server:

```
/B-YEP-410> ./zappy_server -help
USAGE: ./zappy_server -p port -x width -y height -n name1 name2 ... -c clientsNb -f freq
    port        is the port number
    width       is the width of the world
    height      is the height of the world
    nameX       is the name of the team X
    clientsNb   is the number of authorized clients per team
    freq        is the reciprocal of time unit for execution of actions
```

To use Zappy GUI:

```
/B-YEP-410> ./zappy_gui -help
USAGE: ./zappy_ai -p port -h machine
    port        is the port number
    machine     is the name of the machine; localhost by default
```

To use Zappy IA:

```
/B-YEP-410> ./zappy_ai -help
USAGE: ./zappy_ai -p port -n name -h machine
    port        is the port number
    name        is the name of the team
    machine     is the name of the machine; localhost by default
```

4.2 Commands

4.2.1 Description

In order to better define the communication protocol, we have made a summary table of the different commands used for the server transmission and the user interface reception. A small description of each command is listed in the table and all parameters are defined in the index.

4.2.2 Index

- **X** width or horizontal position
- **Y** height or vertical position
- **q0** resource 0 (food) quantity
- **q1** resource 1 (linemate) quantity
- **q2** resource 2 (deraumere) quantity
- **q3** resource 3 (sibur) quantity
- **q4** resource 4 (mendiane) quantity
- **q5** resource 5 (phiras) quantity
- **q6** resource 6 (thystame) quantity
- **n** player number
- **O** orientation: 1(N), 2(E), 3(S), 4(W)
- **L** player or incantation level
- **e** egg number
- **T** time unit
- **N** name of the team
- **R** incantation result
- **M** message
- **I** resource number

4.2.3 Commands

SERVER MESSAGES	GUI COMMANDS	DESCRIPTION
msz X Y	msz X Y	map size
bct X Y q0 q1 q2 q3 q4 q5 q6	bct X Y	content of a tile
bct X Y q0 q1 q2 q3 q4 q5 q6n * nb	mct	content of the map (all the tiles)
tna Nn * nbr_teams	tna	name of all the teams
pnw n X Y O L N		creation of a new player
pns n R		player state (1 alive, 0 not used)
ppo n X Y O	ppo #n	player's position
plv n L	plv #n	player's level
pin n q0 q1 q2 q3 q4 q5 q6	pin #n	player's inventory
pex n		explusion
pbc n M		broadcast
pic X Y L n n …		start of an incantation (by the first player)
pie X Y R		end of an incantation
pdr n i		resource dropping
pgt n i		resource collecting
pdi n		death of a player
enw n e X Y		an egg was laid by a player
eht e		egg hatching
sgt T	sgt	time unit request
seg N		end of game
suc		unknown command
sbp		command parameter
look		look animation
pgte n R		take ressource end 1 or 0
pexed n X Y		ejected to this coords
pnw and pin	pls	get all players at start

4.3 Part 1: Server

4.3.1 Description

The server is the main engine that will interfere between the AI and the GUI. It will exchange all data with the AI and execute these instructions. At the same time, it send the information to the GUI.

4.3.2 Communication

In this part we will see in more details the communication protocol and the management of the commands received by the server.

Connection

As noticed on the diagram on the home page, the server communicates with the AI and the GUI:

- One function takes care of the GUI connection: *gui_command_connection*
- Another one takes care of the AI connection which takes possession of the players: *player_command_connection*.

```
void gui_command_connection(char **args, client_t *client)
{
    for (size_t i = 0; GUI_COMMANDS[i].cmd; i++) {
        if (strcmp(GUI_COMMANDS[i].cmd, args[0]) == 0) {
            enqueue_command(client->player, args, &GUI_COMMANDS[i]);
            break;
        }
    }
}
```

```
void player_command_connection(char **args, client_t *client)
{
    for (size_t i = 0; PLAYER_COMMANDS[i].cmd; i++) {
        if (strcmp(PLAYER_COMMANDS[i].cmd, args[0]) == 0) {
            enqueue_command(client->player, args, &PLAYER_COMMANDS[i]);
            break;
        }
    }
}
```

Commands Queue

In order to receive a large number of commands and to be able to process them all, we have set up a FIFO (First In First Out) type queue system:

```
void enqueue_command(player_t *player, char **args, command_t *command)
{
    int i = 0;
    int j = 0;
    int nb_args = count_args(args);

    for (i = 0; player->command[i]; i++);
    if (i == MAX_COMMAND)
        return;
    player->command[i] = command;
    player->command[i]->args = malloc(sizeof(char *) * (nb_args + 1));
    if (player->command[i]->args == NULL)
        show_error("Malloc fail.");
    for (j = 0; args[j]; j++) {
        player->command[i]->args[j] = strdup(args[j]);
    }
    player->command[i]->args[j] = NULL;
}
```

Here is the dequeue function which allows to exit the chain in order to execute the command entered:

```
void dequeue_command(player_t *player)
{
    free_array(&(player->command[0]->args));
    for (int i = 0; player->command[i]; i++) {
        if (i == MAX_COMMAND) {
            player->command[i] = NULL;
            break;
        }
        player->command[i] = player->command[i + 1];
    }
}
```

4.3.3 Player

We will explain the management of players in our server.

Inventory

The player inventory is composed of two main parts:

- The foods that serve the survival of the player
- The ores which are used to make an incantation

```
typedef struct inventory_s {
    int food;
    int linemate;
    int deraumere;
    int sibur;
    int mendiane;
    int phiras;
    int thystame;
} inventory_t;
```

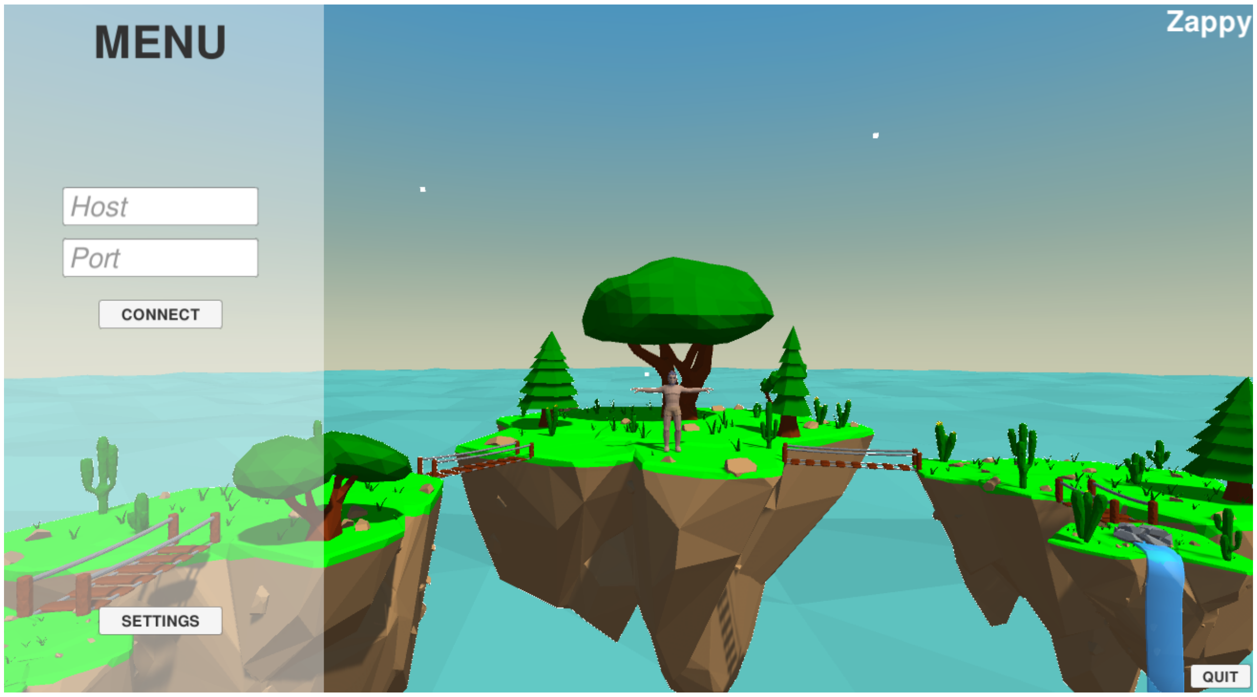
4.4 Part 2: GUI

4.4.1 Description

To have a better vision of the game, which is simple, clear and ergonomic, we made a user interface with the Unity engine. We will describe below how our GUI works.

4.4.2 Main menu

First of all, a main menu will appear. To launch the game you just have to enter the ip of the server and its corresponding port. In this menu, it is also possible to set the keys and the sensitivity.

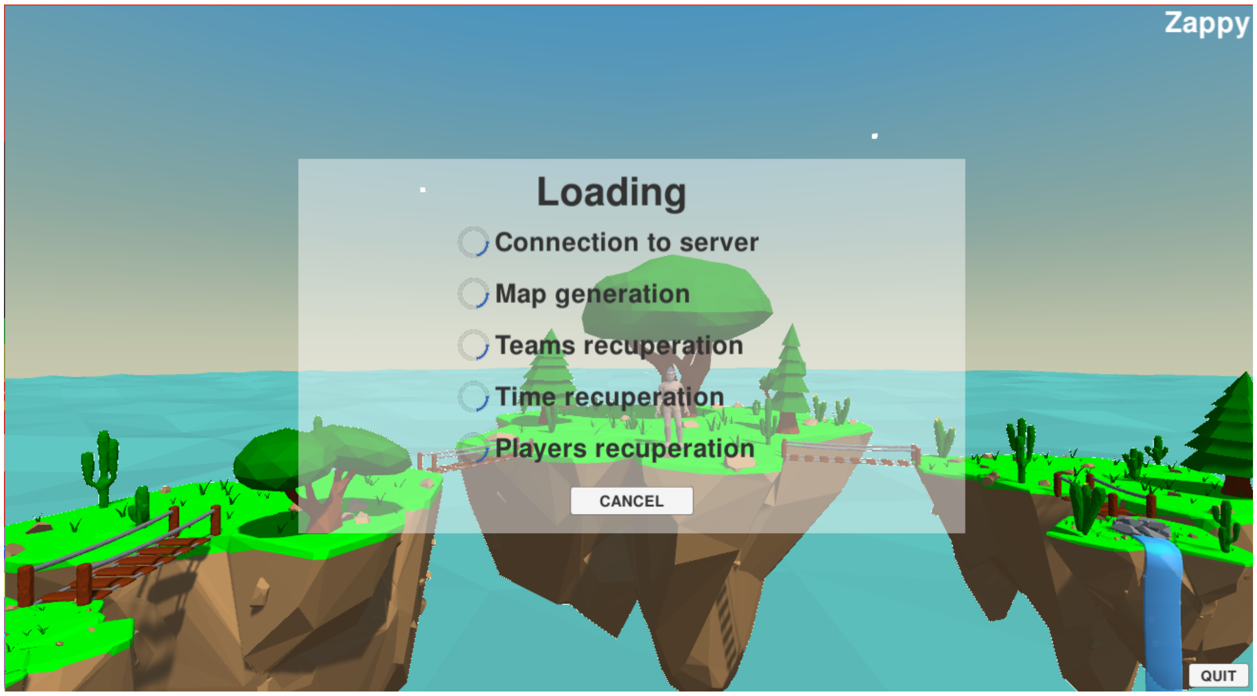


```
// Host field
public void setHostText(string value) {
    host_text.GetComponent<UnityEngine.UI.InputField>().text = value;
    host = value;
}
// Port field
public void setPortText(string value) {
    port_text.GetComponent<UnityEngine.UI.InputField>().text = value;
    port = value;
}
// Connect button
public void click_connect() {
    string host = host_text.GetComponent<UnityEngine.UI.InputField>().text;
    string port = port_text.GetComponent<UnityEngine.UI.InputField>().text;

    connect_menu.GetComponent<Connect_menu>().setInfos(host, int.Parse(port));
    connect_menu.SetActive(true);
}
// Settings button
public void click_settings() {
    host = host_text.GetComponent<UnityEngine.UI.InputField>().text;
    port = port_text.GetComponent<UnityEngine.UI.InputField>().text;
    instance.SetActive(false);
    leave_button.GetComponent<Leave_settings>().main = true;
    settings_menu.SetActive(true);
}
```

4.4.3 Attempt server information

Once the connection information is filled in, the game will wait for information from the server (map size, game time, teams and players, . . .). When all the information sent by the server are received by the game, it will start the game automatically.



```
// Map generation
public void validateMsz() {
    if (validations.Contains("msz"))
        return;
    map_text.GetComponent<Loading_text>().validate();
    validations.Add("msz");
    connection_controller.GetComponent<Connection_manager>().send("tna");
    finalValidation();
}
// Teams recuperation
public void validateTna() {
    if (validations.Contains("tna"))
        return;
    validations.Add("tna");
    teams_text.GetComponent<Loading_text>().validate();
    connection_controller.GetComponent<Connection_manager>().send("sgt");
    finalValidation();
}
// Time recuperation
public void validateSgt() {
    if (validations.Contains("sgt"))
        return;
    validations.Add("sgt");
    time_text.GetComponent<Loading_text>().validate();
    connection_controller.GetComponent<Connection_manager>().send("pls");
```

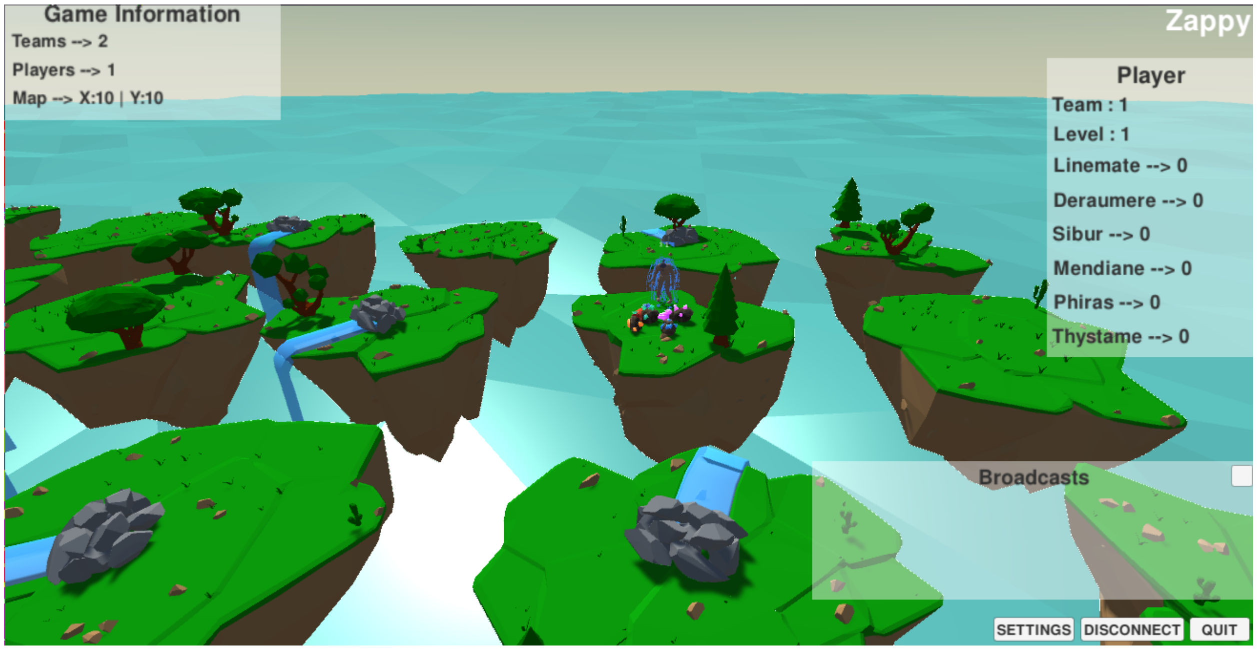
(continues on next page)

(continued from previous page)

```
    finalValidation();
}
// Players recuperation
public void validatePls() {
    if (validations.Contains("pls"))
        return;
    validations.Add("pls");
    pls_text.GetComponent<Loading_text>().validate();
    finalValidation();
}
```

4.4.4 The complete game

In the game, you can see the different islands that represent the tiles of the map. Each island contains food and minerals. The players move between the islands to collect resources. In real time, the information of the map are displayed, as well as the broadcast of the players that can be deciphered. By clicking on the islands or players, we can see what they contain (resources, inventory, . . .)



Game Information
Teams --> 2
Players --> 1
Map --> X:10 | Y:10

Zappy

Player
Team : 1
Level : 1
Linemate --> 0
Deraumere --> 0
Sibur --> 0
Mendiane --> 0
Phiras --> 0
Thystame --> 0

Note: The gui is in permanent waiting of a command, once this one received, it will compare it in the file **Args_manager.cs** and execute the action in the **Commands.cs** script.

4.5 Part 3: IA

4.5.1 Description

Each player of the game is by default inactive, to animate a player the artificial intelligence must take possession of it. To do this, it must communicate with the server and send it precise instructions. **Its objective:** reach level 8 as soon as possible!

4.5.2 Initialisation

Client interface

Here is Client class constructor:

```
self.team = team_name
self.hostname = hostname
self.port = port
self.client_num = 0
self.data = Data(0, 0)
self.socket = None
self.selectors = selectors.DefaultSelector()
self.just_log = 0
self.logged = 0
self.ia = IA(self.team)
```

Server Connection

Initialise a socket which connects to the server. We initialize the selectors to and be sure to receive the Welcome message.

```
def connect_to_server(self):
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket.setblocking(False)
    self.socket.connect_ex((self.hostname, safe_cast(self.port, int)))
    events = selectors.EVENT_READ | selectors.EVENT_WRITE
    self.selectors.register(self.socket, events)
```

4.5.3 Algorithm

The AI engine is based on a shared inventory algorithm that allows all players in the team to know the resources needed for the incantation. Below is the main source code of the algorithm:

```
def algorithm(self):
    if self.step == -1:
        self.data_to_write = "Connect_nbr\n"
        self.step += 1
    elif self.step == 0:
        if self.commands_list:
            self.data_to_write = self.commands_list[0]
```

(continues on next page)

(continued from previous page)

```
            self.commands_list = self.commands_list[1:]
        else:
            self.data_to_write = "Inventory\n"
            self.step += 1
    elif self.step == 1:
        if self.new_object:
            if not self.check_incantation():
                message = bytes(self.sxor(self.team, ("inventory" + str(self.client_num)␣
↪+ ";" + str(self.level) + ";" + str(json.dumps(self.inventory)))), "utf-8").hex()
                self.data_to_write = "Broadcast " + message + "\n"
            else:
                message = bytes(self.sxor(self.team, (str(self.client_num) + ";
↪incantation;" + str(self.level))), "utf-8").hex()
                self.data_to_write = "Broadcast " + message + "\n"
                self.master_incantation = 1
                self.step = 4
                self.incantation = 1
                self.new_object = False
                return
            self.new_object = False
        else:
            self.step += 1
            self.algorithm()
            return
        self.step += 1
    elif self.step == 2:
        if "food" in self.inventory and self.inventory["food"] < 15:
            self.data_to_write = "Look\n"
        else:
            self.step += 1
            self.algorithm()
            return
        self.step += 1
    elif self.step == 3:
        if "food" in self.inventory and self.inventory["food"] < 15:
            self.commands_list = self.parse_look(self.look, "food")
            self.step = 0
        else:
            self.to_search = self.search_good_ressources()
            self.commands_list = self.parse_look(self.look, self.to_search)
            self.step = 0
    elif self.step == 4:
        if self.incantation == 0:
            data = bytes(self.sxor(self.team, str(self.client_num) + " on my way"), "utf-
↪8").hex()
            self.data_to_write = "Broadcast " + data + "\n"
            self.incantation = 1
            return
        if self.master_incantation >= 6:
            self.step += 1
            return
        if self.master_incantation >= 1:
```

(continues on next page)

(continued from previous page)

```
            pass
        elif self.commands_list and not self.ready_for_incantation:
            self.data_to_write = self.commands_list[0]
            self.commands_list = self.commands_list[1:]
            self.clear_broadcast = 1
            if not self.commands_list:
                self.clear_read = 1
        elif self.ready_for_incantation and "Broadcast" in self.data_to_write:
            self.step += 1
        else:
            self.data_to_write = ""
    elif self.step == 5:
        self.clear_broadcast = 0
        self.data_to_write = "Look\n"
        self.step += 1
    elif self.step == 6:
        if self.master_incantation >= 6:
            self.start_incantation()
        if self.step != 7:
            self.drop_object_incantation()
            if self.commands_list:
                self.data_to_write = self.commands_list[0]
                self.commands_list = self.commands_list[1:]
            else:
                self.data_to_write = "Inventory\n"
    elif self.step == 7:
        if self.master_incantation < 6:
            self.data_to_write = ""
            return
        if self.commands_list:
                print(self.level, " ", self.look)
                self.data_to_write = self.commands_list[0]
                self.commands_list = self.commands_list[1:]
        else:
            self.commands_list = ["Inventory\n"]
            self.commands_list = ["Look\n"]
    elif self.step == 8:
        self.data_to_write = ""
```