

jun 27, 17 14:46

**Thread.h**

Page 1/1

```
1  #ifndef Z_TPGRUPAL_THREAD_H
2  #define Z_TPGRUPAL_THREAD_H
3
4
5  #include <thread>
6
7  class Thread {
8  protected:
9      std::thread thread;
10
11 public:
12     void start();
13
14     virtual void run() = 0;
15
16     void join();
17 };
18
19
20 #endif //Z_TPGRUPAL_THREAD_H
```

jun 27, 17 14:46

**Thread.cpp**

Page 1/1

```
1  #include "Thread.h"
2
3  void Thread::start() {
4      thread = std::thread(&Thread::run, this);
5  }
6
7  void Thread::join() {
8      this->thread.join();
9  }
```

jun 27, 17 14:46

test\_client.cpp

Page 1/1

```

1  #include "socket.h"
2  #include "messenger.h"
3  #include <mutex>
4  #include <iostream>
5  #include "clientReceiverTest.h"
6
7  int main() {
8      Socket s("127.0.0.1", 8000);
9      Messenger m(s);
10     std::mutex mutex;
11     clientReceiverTest client(m,mutex);
12     client.start();
13
14     while(true) {
15         std::cout << "Enter message: " << std::endl << ">> ";
16         std::string msg;
17         std::getline(std::cin, msg);
18         m.sendMessage(msg);
19     }
20 }

```

jun 27, 17 14:46

socket.h

Page 1/1

```

1  #ifndef TP3TALLER_COMMON_SOCKET_H
2  #define TP3TALLER_COMMON_SOCKET_H
3
4  #include <string>
5  #include "socketError.h"
6
7  #define LISTEN_BACKLOG 10 // Amt. of connections to have in the accept backlog
8
9  // Socket class. Wraps functionality of glibc's socket functions.
10 class Socket {
11     int fd;
12
13 public:
14     /* Server constructor. Creates a socket, binds and listens to the specified
15      * port. */
16     explicit Socket(int port);
17
18     /* Client constructor. Creates a socket and attempts to connect to the
19      * specified address/port. Raises exception if the connection fails. */
20     Socket(const char *addr, int port);
21
22     ~Socket();
23
24     // Returns a new client.
25     Socket accept_client();
26
27     // Sends/recieves len bytes of data
28     ssize_t send(const char *msg, unsigned int len);
29     ssize_t receive(char *dest, size_t len);
30
31     // Wrapper for socket shutdown/close
32     void shutdown();
33     void close(); // Effectively makes the socket object useless
34
35     bool is_valid();
36     // Move constructor
37     Socket(Socket& other);
38
39 private:
40     Socket();
41
42     Socket(Socket&) = delete;
43     void operator=(Socket&) = delete;
44 };
45
46 #endif //TP3TALLER_COMMON_SOCKET_H

```

jun 27, 17 14:46

socketError.h

Page 1/1

```

1  //
2  // Created by rodian on 22/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_SOCKETERROR_H
6  #define Z_TPGRUPAL_SOCKETERROR_H
7
8  #include <iostream>
9  #include <cstring>
10
11  //////////////////////////////////
12  // SocketError Class to warn
13  // of an error on the socket
14  //////////////////////////////////
15
16  class SocketError : public std::exception {
17  private:
18      char buffer[124];
19
20  public:
21      explicit SocketError(const char* message, ...) noexcept;
22
23      // Returns the error message
24      virtual const char* what() const noexcept;
25  };
26
27  #endif //Z_TPGRUPAL_SOCKETERROR_H

```

jun 27, 17 14:46

socketError.cpp

Page 1/1

```

1  //
2  // Created by rodian on 22/05/17.
3  //
4
5  #include "socketError.h"
6
7  SocketError::SocketError(const char *message, ...) noexcept {
8      strncpy(buffer, message, strlen(message));
9  }
10
11  const char* SocketError::what() const noexcept {
12      return buffer;
13  }

```

jun 27, 17 14:46

socket.cpp

Page 1/2

```

1  #include <sys/socket.h>
2  #include <unistd.h>
3  #include <netinet/in.h>
4  #include <arpa/inet.h>
5  #include <cstring>
6  #include <utility>
7  #include <iostream>
8  #include <string>
9  #include "socket.h"
10
11 Socket::Socket(int port) {
12     fd = socket(AF_INET, SOCK_STREAM, 0);
13     if (fd < 0) {
14         throw SocketError("Couldn't create a socket!\n");
15     }
16
17     struct sockaddr_in srv;
18     memset(&srv, 0, sizeof(srv));
19     srv.sin_family = AF_INET;
20     srv.sin_addr.s_addr = htonl(INADDR_ANY);
21     srv.sin_port = htons((uint16_t) port);
22
23     int error = bind(fd, (struct sockaddr *) &srv, sizeof(srv));
24     if (error) {
25         throw SocketError("Error binding socket on creation! "
26                             "Most likely port already in use");
27     }
28
29     listen(fd, LISTEN_BACKLOG);
30 }
31
32 Socket::Socket(const char *addr, int port) {
33     fd = socket(AF_INET, SOCK_STREAM, 0);
34     struct sockaddr_in srv;
35     srv.sin_family = AF_INET;
36     srv.sin_port = htons((uint16_t)port);
37     srv.sin_addr.s_addr = inet_addr(addr);
38
39     socklen_t len = (socklen_t)sizeof(struct sockaddr);
40     int error = connect(fd, (struct sockaddr *) &srv, len);
41
42     if (error) {
43         throw SocketError("Error connecting to server!");
44     }
45 }
46
47 Socket Socket::accept_client() {
48     struct sockaddr_in client;
49     socklen_t cliLen = (socklen_t) sizeof(struct sockaddr_in);
50
51     int client_fd = accept(fd, (struct sockaddr *) &client, &cliLen);
52     if (client_fd < 0 || fd < 0) {
53         throw SocketError("Socket disconnected");
54     }
55
56     Socket client_socket;
57     client_socket.fd = client_fd;
58     return client_socket;
59 }
60
61 Socket::Socket() {
62 }
63
64 Socket::Socket(Socket& other) {
65     fd = other.fd;
66

```

jun 27, 17 14:46

socket.cpp

Page 2/2

```

67     other.fd = -1; // "Deactivates" other
68 }
69
70 ssize_t Socket::send(const char *msg, unsigned int len) {
71     size_t total_bytes = 0;
72     ssize_t sent = 1;
73
74     // Sends msg until it's complete OR socket_send returns 0 (connection
75     // closed)
76     while (total_bytes < len ^ sent) {
77         sent = ::send(fd, msg + total_bytes, len - total_bytes,
78                     MSG_NOSIGNAL);
79         if (sent < 0) {
80             return -1;
81         }
82         total_bytes += sent;
83     }
84
85     return total_bytes;
86 }
87
88 ssize_t Socket::receive(char *dest, size_t len) {
89     ssize_t received = 1;
90     size_t total_bytes = 0;
91
92     // Writes to dest until it's complete OR socket_recv returns 0 (connection
93     // closed)
94     while (total_bytes < len ^ received) {
95         received = recv(fd, dest + total_bytes, len - total_bytes,
96                     MSG_NOSIGNAL);
97         if (received < 0) {
98             return -1;
99         }
100         total_bytes += received;
101     }
102     return total_bytes;
103 }
104
105 Socket::~Socket() {
106     if (fd > 0) {
107         close();
108     }
109 }
110
111 void Socket::shutdown() {
112     ::shutdown(fd, SHUT_RDWR);
113     fd = -1;
114 }
115
116 void Socket::close() {
117     ::close(fd);
118     fd = -1;
119 }
120
121 bool Socket::is_valid() {
122     return fd > 0;
123 }
124

```

jun 27, 17 14:46

messenger.h

Page 1/1

```

1 //
2 // Created by rodian on 22/05/17.
3 //
4
5 #ifndef Z_TPGRUPAL_MESSENGER_H
6 #define Z_TPGRUPAL_MESSENGER_H
7
8 #include "socket.h"
9 #include <iostream>
10 #include <string>
11 //////////////////////////////////////////////////
12 // Messenger Class meant to use sockets
13 // to send messages between Client and
14 // Server using a specific protocol.
15 // Send lenght of message first, then the message.
16 //////////////////////////////////////////////////
17 class Messenger{
18 private:
19     Socket socket;
20     bool connected;
21
22 public:
23     // Recieves a unique socket to send
24     // and recieves messages from
25     explicit Messenger(Socket& socket);
26
27     // Recieves a Message from the remote
28     // connected socket.
29     // Returns the message on a string
30     std::string recieveMessage();
31
32     // Sends a message to the remote socket
33     // Recieves the message on a string
34     void sendMessage(const std::string &message);
35
36     // Shuts down the socket for read and write
37     void shutdown();
38
39     // If the sockets are still connected returns true
40     // otherwise false.
41     bool isConnected();
42
43     ~Messenger();
44 };
45
46 #endif //Z_TPGRUPAL_MESSENGER_H

```

jun 27, 17 14:46

messenger.cpp

Page 1/1

```

1 #include <netinet/in.h>
2 #include <string>
3 #include "socket.h"
4 #include "messenger.h"
5
6 Messenger::Messenger(Socket& socket) : socket(std::move(socket)),
7                                     connected(true) {}
8
9 #define MSG_SIZE 1024
10
11 std::string Messenger::recieveMessage() {
12     // Receive length first, then the message
13     uint32_t len = 0;
14     socket.receive((char*) &len, sizeof(len));
15     len = ntohl(len);
16     char* buf = new char[len];
17     ssize_t sent = socket.receive(buf, len);
18     if (sent <= 0) {
19         throw(SocketError("Socket closed"));
20     }
21     std::string result(buf);
22
23     delete[] buf;
24     return result;
25 }
26
27 void Messenger::sendMessage(const std::string &message) {
28     if (this->connected) {
29         uint32_t len = (uint32_t) message.size() + 1;
30         // Send length first, then the message
31         uint32_t network_len = htonl(len);
32         socket.send((char *) &network_len, sizeof(network_len));
33         socket.send(message.c_str(), len);
34     }
35 }
36
37 void Messenger::shutdown() {
38     socket.shutdown();
39 }
40
41 bool Messenger::isConnected() {
42     return socket.is_valid();
43 }
44
45 Messenger::~Messenger() {
46 }

```

jun 27, 17 14:46

**clientReceiverTest.h**

Page 1/1

```

1  //
2  // Created by rodian on 15/06/17.
3  //
4
5  #ifndef Z_TPGRUPAL_CLIENTRECEIVERTEST_H
6  #define Z_TPGRUPAL_CLIENTRECEIVERTEST_H
7
8
9  #include <mutex>
10 #include "messenger.h"
11 #include "Thread.h"
12
13 class clientReceiverTest: public Thread {
14 private:
15     std::mutex &m;
16     Messenger &messenger;
17     bool listen;
18 public:
19     clientReceiverTest(Messenger &messenger, std::mutex& m);
20
21     void run();
22 };
23
24
25 #endif //Z_TPGRUPAL_CLIENTRECEIVERTEST_H

```

jun 27, 17 14:46

**clientReceiverTest.cpp**

Page 1/1

```

1  //
2  // Created by rodian on 15/06/17.
3  //
4
5  #include "clientReceiverTest.h"
6
7  clientReceiverTest::clientReceiverTest(Messenger &messenger, std::mutex &m) :
8      m(m), messenger(messenger), listen(true) {}
9
10 void clientReceiverTest::run() {
11     while (listen) {
12         std::string msg = messenger.receiveMessage();
13
14         //         m.lock();
15         std::cerr << "Answer: " << msg << std::endl;
16         //         m.unlock();
17     }
18 }

```

jun 27, 17 14:46

weapon.h

Page 1/1

```

1  //
2  // Created by rodian on 21/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_WEAPON_H
6  #define Z_TPGRUPAL_WEAPON_H
7
8  #include <iostream>
9  #include <vector>
10 #include "size.h"
11 #include "Occupant.h"
12 #include "bullet.h"
13
14 class Weapon {
15 private:
16     std::string type;
17     int damage, w_speed;
18     bool explosive;
19     Size w_size;
20     Bullet bullet;
21
22 public:
23     Weapon(std::string& type, int damage, int w_speed, bool explosive,
24           Size w_size);
25
26     void setNewTarget(Occupant* target);
27
28     Bullet* shotTarget(Occupant* target);
29
30     bool isTheAttackExplosive();
31
32     std::vector<Position>& getBulletRoad();
33
34     Size getBulletSize() const;
35
36     void recalculateRoadToTarget();
37
38     void movePosition(int x,int y);
39 };
40
41 #endif //Z_TPGRUPAL_WEAPON_H

```

jun 27, 17 14:46

weapon.cpp

Page 1/1

```

1  //
2  // Created by rodian on 21/05/17.
3  //
4
5  #include "weapon.h"
6
7  Weapon::Weapon(std::string& type, int dmg, int w_speed,
8                bool explosive, Size w_size) : type(type), damage(dmg),
9                w_speed(w_speed), explosive(explosive), w_size(w_size),
10                bullet(type,dmg,w_speed,w_size) {}
11
12  Bullet* Weapon::shotTarget(Occupant* target) {
13      Bullet* shoted_bullet = new Bullet(type,damage,w_speed,w_size,target);
14      shoted_bullet->shotTarget(target);
15      return shoted_bullet;
16  }
17
18  bool Weapon::isTheAttackExplosive() {
19      return explosive;
20  }
21
22  void Weapon::setNewTarget(Occupant* target) {
23      this->bullet.shotTarget(target);
24  }
25
26  std::vector<Position>& Weapon::getBulletRoad() {
27      return this->bullet.getRoad();
28  }
29
30  Size Weapon::getBulletSize() const {
31      return this->w_size;
32  }
33
34  void Weapon::recalculateRoadToTarget() {
35      this->bullet.calculateRoadToTarget();
36  }
37
38  void Weapon::movePosition(int x, int y) {
39      w_size.moveTo(x,y);
40      this->bullet.setStartLocation(x,y);
41  }
42

```

jun 27, 17 14:46

unitMold.h

Page 1/1

```

1 //
2 // Created by rodian on 22/05/17.
3 //
4
5 #ifndef Z_TPGRUPAL_UNITMOLD_H
6 #define Z_TPGRUPAL_UNITMOLD_H
7
8 #include "unit.h"
9
10 class UnitMold {
11 private:
12     int tec_level, life, range, width, height, unit_speed, fire_rate,
13         creation_time, creation_quantity;
14     std::string type, weapon_type;
15
16 public:
17     UnitMold(int tec_level, int life, int range, int width, int height,
18             int unit_speed, int fire_rate, int creation_time,
19             int creation_quantity, std::string &type,
20             std::string &weapon_type);
21
22     Unit *createUnit(int id, Size u_size, Map& map,
23                     Weapon &weapon);
24
25     // Returns the technology level needed to create Tough unit
26     int getTechnologyLevel() const;
27
28     // Returns creation time in seconds
29     int getCreationTime();
30
31     // Returns the type of unit that this mold creates
32     std::string getTypeOfUnit() const;
33
34     std::string getWeaponType() const;
35
36     Size getUnitSize();
37
38     int getCreationQuantity() const;
39
40     int getFireRate() const;
41
42     int getLife() const;
43
44 };
45
46 #endif //Z_TPGRUPAL_UNITMOLD_H

```

jun 27, 17 14:46

unitMold.cpp

Page 1/1

```

1 //
2 // Created by rodian on 22/05/17.
3 //
4
5 #include "unitMold.h"
6
7 UnitMold::UnitMold(int tec_level, int life, int range, int width, int height,
8                   int unit_speed, int fire_rate, int time, int creation_quantit
9                   Y,
10                   std::string &type, std::string &weapon_type) :
11     tec_level(tec_level), life(life), range(range), width(width), height(height),
12     unit_speed(unit_speed), fire_rate(fire_rate), creation_time(time),
13     creation_quantity(creation_quantity), type(type), weapon_type(weapon_type) {}
14
15 Unit *UnitMold::createUnit(int id, Size u_size, Map& map,
16                             Weapon &weapon) {
17     Position u_pos = u_size.getPosition();
18     Compass* compass = new Compass(map, u_size, id, unit_speed);
19     // get closest valid position from fabric
20     Position valid_pos = compass->getAValidPositionForDestiny(u_pos);
21     u_size.moveTo(valid_pos.getX(), valid_pos.getY());
22     int x_range = valid_pos.getX() - range;
23     int y_range = valid_pos.getY() - range;
24     int w_range = range * 2 + width;
25     int h_range = range * 2 + height;
26     Size unit_range(x_range, y_range, w_range, h_range);
27
28     Unit* new_unit = new Unit(id, life, type, unit_speed, u_size, unit_range,
29                               compass, weapon, fire_rate);
30     return new_unit;
31 }
32
33 int UnitMold::getTechnologyLevel() const {
34     return tec_level;
35 }
36
37 int UnitMold::getCreationTime() {
38     return creation_time;
39 }
40
41 std::string UnitMold::getTypeOfUnit() const {
42     return type;
43 }
44
45 Size UnitMold::getUnitSize() {
46     return Size(0, 0, width, height);
47 }
48
49 int UnitMold::getCreationQuantity() const {
50     return creation_quantity;
51 }
52
53 std::string UnitMold::getWeaponType() const {
54     return this->weapon_type;
55 }
56
57 int UnitMold::getFireRate() const {
58     return fire_rate;
59 }
60
61 int UnitMold::getLife() const {
62     return life;
63 }

```



jun 27, 17 14:46	unit.h	Page 1/2
<pre> 1  // 2  // Created by rodian on 18/05/17. 3  // 4 5  #ifndef Z_TPGRUPAL_UNIT_H 6  #define Z_TPGRUPAL_UNIT_H 7 8  #include "Occupant.h" 9  #include "compass.h" 10 #include "teamable.h" 11 #include "bullet.h" 12 #include "weapon.h" 13 14 #define ATKSTATE "atk" 15 #define MOVESTATE "mv" 16 #define STANDINGSTATE "std" 17 #define GRABBINGSTATE "grb" 18 #define FLAGTYPE "flag" 19 #define GRUNTTYPE "grunt" 20 21 22 class Unit: public Occupant { 23 private: 24     Compass* compass; 25     Weapon weapon; 26     int unit_speed, fire_rate, fire_count; 27     std::string state, action; 28     Size range, grab_range; 29     std::vector&lt;Position&gt; road; 30     Occupant* target; 31     Teamable* grab_target; 32     std::vector&lt;Bullet*&gt; bullets; 33     bool got_target, mount_vehicle; 34 35 public: 36     Unit(int id, int life, std::string type, int unit_speed, Size size, 37         Size range, Compass* compass, Weapon &amp;weapon, int fire_rate); 38 39     // This method is the one who makes the unit make the action that he must 40     // do depending on his state 41     void makeAction(); 42 43     // Calculates the minimum road to destiny (x,y) 44     void calculateRoadTo(int x, int y); 45 46     // Returns the current position of the unit 47     Position getCurrentPosition() const; 48 49     // Returns "atk" if is attacking, "mv" if is moving, "std" if 50     // is standing still 51     std::string getActionState() const; 52 53     void grab(); 54 55     void setTargetToAttack(Occupant* target); 56 57     void setTargetToGrab(Teamable* object, std::string type); 58 59     bool doYouHaveAnyBullets(); 60 61     std::vector&lt;Bullet*&gt; collectBullets(); 62 63     bool checkIfTargetIsOnRange(); 64 65     // The bullet will hit if there is no Occupant in the middle. 66     // except for bridges </pre>		

jun 27, 17 14:46	unit.h	Page 2/2
<pre> 67     bool checkIfBulletWillHit(std::vector&lt;Position&gt;&amp; b_road, Size&amp; b_size); 68 69     void getOnRangeOf(int x, int y); 70 71     bool checkIfAlreadyOnMyWay(int x, int y); 72 73     Size getNextPosition(int steps); 74 75     void recalculateMyStartPosition(); 76 77     ~Unit(); 78 79 private: 80     // Indicates the Unit to make the next step on the road. 81     // Make sure of use the calculateRoadTo method before this one. 82     void move(); 83 84     // this method creates the bullets to attack a certain target 85     void attack(); 86 87     bool onRangeToGrabTarget(); 88 }; 89 90 91 #endif //Z_TPGRUPAL_UNIT_H </pre>		

jun 27, 17 14:46

unit.cpp

Page 1/5

```

1  //
2  // Created by rodian on 18/05/17.
3  //
4
5  #include "unit.h"
6  #define NEUTRAL "Neutral"
7  Unit::Unit(int id, int life, std::string type, int unit_speed, Size size,
8             Size range, Compass* compass, Weapon &weapon, int fire_rate) :
9      Occupant(id, life, type, size), compass(compass), weapon(weapon),
10     unit_speed(unit_speed), fire_rate(fire_rate), fire_count(0),
11     state(STANDINGSTATE), action(STANDINGSTATE), range(range),
12     grab_range(size.getPosition().getX() - 1, size.getPosition().getY() - 1,
13               size.getWidth() + 2, size.getHeight() + 2),
14     target(this), grab_target(this), got_target(false),
15     mount_vehicule(false) {
16     compass->changeUnitId(id);
17     compass->buildNodeMap();
18 }
19
20 void Unit::makeAction() {
21     if (this->state == STANDINGSTATE) {
22         if (this->team != NEUTRAL) {
23             if (!got_target) {
24                 // Check for enemies around you. If so, state = ATKSTATE
25                 this->changed = false;
26                 target = compass->checkForEnemiesOnRange
27                     (*(Occupant *) this, range);
28                 if (target->getId() != this->id) {
29                     got_target = true;
30                 }
31             } else {
32                 if (target->areYouAlive() ^ checkIfTargetIsOnRange()) {
33                     attack();
34                     this->action = ATKSTATE;
35                 } else {
36                     got_target = false;
37                     this->action = STANDINGSTATE;
38                     this->changed = true;
39                 }
40             }
41         }
42     }
43     if (this->state == MOVESTATE) {
44         this->move();
45         if (road.empty()) {
46             this->state = STANDINGSTATE;
47             this->action = STANDINGSTATE;
48             this->changed = true;
49         }
50     }
51     if (this->state == ATKSTATE) {
52         if (target->areYouAlive()) {
53             if (checkIfTargetIsOnRange()) {
54                 if (!road.empty())
55                     road.clear();
56                 attack();
57             } else {
58                 // If target is not on range move till it is
59                 // calculate road to target
60                 if (road.empty()) {
61                     Position trg_pos = target->getPosition();
62                     getOnRangeOf(trg_pos.getX(), trg_pos.getY());
63                 }
64                 move();
65             }
66         } else {

```

jun 27, 17 14:46

unit.cpp

Page 2/5

```

67         this->state = STANDINGSTATE;
68         this->action = STANDINGSTATE;
69         this->changed = true;
70     }
71 }
72 if (this->state == GRABBINGSTATE) {
73     if (this->action != MOVESTATE) {
74         Position target_pos = grab_target->getPosition();
75         if (!road.empty())
76             road.clear();
77         Position actual = obj_size.getPosition();
78         road = compass->getFastestWay(actual, target_pos);
79         this->action = MOVESTATE;
80     } else {
81         if (onRangeToGrabTarget()) {
82             grab();
83         } else if (road.empty()) {
84             this->state = STANDINGSTATE;
85             this->action = STANDINGSTATE;
86             this->changed = true;
87             mount_vehicule = false;
88         } else if (!road.empty()) {
89             move();
90         }
91     }
92 }
93 }
94
95 void Unit::calculateRoadTo(int x, int y) {
96     if (!checkIfAlreadyOnMyWay(x, y)) {
97         this->state = MOVESTATE;
98         Position destination(x, y);
99         Position actual = obj_size.getPosition();
100         road = compass->getFastestWay(actual, destination);
101     }
102 }
103
104 void Unit::getOnRangeOf(int x, int y) {
105     Position destination(x, y);
106     Position actual = obj_size.getPosition();
107     road = compass->getFastestWay(actual, destination);
108     Position new_dest = road.back();
109     if (new_dest.getX() == actual.getX() ^ new_dest.getY() == actual.getY()) {
110         this->state = STANDINGSTATE;
111         this->action = STANDINGSTATE;
112     }
113 }
114
115 void Unit::move() {
116     int distance = unit_speed;
117     int steps = 0;
118     bool crash = false;
119     compass->clearCompass();
120
121     while (!road.empty() ^ steps <= distance ^ !crash) {
122         Position pos = road.back();
123         Size next_pos(pos.getX(), pos.getY(),
124                     obj_size.getWidth(), obj_size.getHeight());
125         if (compass->canIWalkToThisPosition(next_pos)) {
126             double t_factor = compass->getTerrainFactorOn(
127                                     pos.getX(), pos.getY());
128             // move unit position, range and weapon
129             this->obj_size.moveTo(pos.getX(), pos.getY());
130             this->weapon.movePosition(pos.getX(), pos.getY());
131             int x_range = pos.getX() -
132                 (range.getWidth() - obj_size.getWidth()) / 2;

```

jun 27, 17 14:46

unit.cpp

Page 3/5

```

133     int y_range = pos.getY() -
134         (range.getHeight() - obj_size.getHeight()) / 2;
135     this->range.moveTo(x_range,y_range);
136     this->grab_range.moveTo(pos.getX() - 1,pos.getY() - 1);
137     this->changed = true;
138     this->action = MOVESTATE;
139     road.pop_back();
140
141     // increase or decrease distance til steps are more than unit speed
142     if (steps < 1) {
143         distance = (int) (t_factor * distance);
144     } else if (steps < 1 ^ unit_speed > 4) {
145         distance = (int) (t_factor * distance *
146             (1-(damage_recv/life_points)));
147     }
148     ++steps;
149 } else {
150     crash = true;
151 }
152 }
153 if (crash) {
154     Position destiny = road.front();
155     Position actual = obj_size.getPosition();
156     road = compass->getFastestWay(actual,destiny);
157     this->action = STANDINGSTATE;
158     this->changed = true;
159 }
160 }
161
162 void Unit::attack() {
163     if (fire_count == 0 ^ fire_count == fire_rate) {
164         fire_count = 0;
165         // make a shot
166         bullets.push_back(weapon.shotTarget(target));
167         this->action = ATKSTATE;
168         this->changed = true;
169         fire_count = 0;
170     }
171     fire_count += 1;
172 }
173
174 std::string Unit::getActionState() const {
175     return this->action;
176 }
177
178 Position Unit::getCurrentPosition() const {
179     return this->obj_size.getPosition();
180 }
181
182 void Unit::setTargetToGrab(Teamable *object, std::string type) {
183     // if its a flag any unit can grab it
184     if (type == FLAGTYPE) {
185         grab_target = object;
186         this->state = GRABBINGSTATE;
187     } else if (this->type == GRUNTTYPE ^ object->getTeam() == NEUTRAL
188         ^ compass->checkIfItIsGrabbable(type) ^ type != FLAGTYPE) {
189         // Only Grunt robots can drive
190         // If is not a flag, is a vehicle
191         grab_target = object;
192         this->state = GRABBINGSTATE;
193         mount_vehicule = true;
194     }
195 }
196
197 void Unit::grab() {
198     this->state = STANDINGSTATE;

```

jun 27, 17 14:46

unit.cpp

Page 4/5

```

199     this->action = STANDINGSTATE;
200     this->changed = true;
201     grab_target->changeTeam(this->team);
202     if (mount_vehicule) {
203         this->damage_recv = this->life_points;
204         mount_vehicule = false;
205     }
206 }
207
208 void Unit::setTargetToAttack(Occupant* target) {
209     std::string type = target->getType();
210     if (!compass->checkIfItIsABuilding(type)) {
211         this->state = ATKSTATE;
212         this->target = target;
213         // clean bullets on weapon when a new target is set
214         this->weapon.setNewTarget(target);
215     } else {
216         if (weapon.isTheAttackExplosive()) {
217             this->state = ATKSTATE;
218             this->target = target;
219             // clean bullets on weapon when a new target is set
220             this->weapon.setNewTarget(target);
221         }
222     }
223 }
224
225 std::vector<Bullet*> Unit::collectBullets() {
226     std::vector<Bullet*> tmp = bullets;
227     bullets.clear();
228     return tmp;
229 }
230
231 bool Unit::checkIfTargetIsOnRange() {
232     bool on_range = true;
233     Size trg_size = target->getSize();
234     if (!range.isThereACollision(trg_size))
235         on_range = false;
236     else {
237         weapon.recalculateRoadToTarget();
238         std::vector<Position>& bullet_road = weapon.getBulletRoad();
239         Size b_size = weapon.getBulletSize();
240         if (!checkIfBulletWillHit(bullet_road,b_size))
241             on_range = false;
242     }
243
244     return on_range;
245 }
246
247 bool Unit::checkIfBulletWillHit(std::vector<Position>& b_road, Size &b_size) {
248     bool will_hit = true;
249     for (auto x: b_road) {
250         b_size.moveTo(x.getX(),x.getY());
251         if (!compass->canBulletWalkToThisPosition(b_size,*this,*target))
252             will_hit = false;
253     }
254     return will_hit;
255 }
256
257 bool Unit::doYouHaveAnyBullets() {
258     return (!bullets.empty());
259 }
260
261 bool Unit::checkIfAlreadyOnMyWay(int x, int y) {
262     bool on_my_way = false;
263     if (state == MOVESTATE) {
264         Position destiny = road.front();

```

jun 27, 17 14:46

unit.cpp

Page 5/5

```

265         if (destiny.getX() == x ^ destiny.getY() == y)
266             on_my_way = true;
267     }
268     return on_my_way;
269 }
270
271 Size Unit::getNextPosition(int steps) {
272     Position dest = road.front();
273     Position pos = road.back();
274
275     while ((dest.getX() != pos.getX() || dest.getY() != pos.getY())
276            ^ steps != 0) {
277         road.pop_back();
278         pos = road.back();
279         --steps;
280     }
281
282     return Size(pos.getX(), pos.getY(), obj_size.getWidth(), obj_size.getHeight());
283 }
284
285 bool Unit::onRangeToGrabTarget() {
286     Size trg_size = grab_target->getSize();
287     return grab_range.isThereACollision(trg_size);
288 }
289
290 void Unit::recalculateMyStartPosition() {
291     Position actual = getPosition();
292     Position valid_pos = compass->getAValidPositionForDestiny(actual);
293     this->obj_size.moveTo(valid_pos.getX(), valid_pos.getY());
294     this->range.moveTo(valid_pos.getX(), valid_pos.getY());
295     this->weapon.movePosition(valid_pos.getX(), valid_pos.getY());
296 }
297
298 Unit::~Unit() {
299     target = nullptr;
300     grab_target = nullptr;
301     for(auto& b: bullets) {
302         delete(b);
303     }
304     delete(compass);
305 }
306

```

jun 27, 17 14:46

territory.h

Page 1/1

```

1  //
2  // Created by rodian on 10/06/17.
3  //
4
5  #ifndef Z_TPGRUPAL_TERRITORY_H
6  #define Z_TPGRUPAL_TERRITORY_H
7
8
9  #include <vector>
10 #include "factory.h"
11 #include "teamable.h"
12
13 class Territory: public Teamable {
14 private:
15     std::map<int, Factory*> factories;
16     Teamable flag;
17     Size territory_size;
18     int id;
19
20 public:
21     Territory(const std::map<int, Factory *> &factories, Position flag_position,
22              Size territory_size, int id);
23
24     void grabFlag(std::string& new_team);
25
26     int getTechLevel();
27
28     void changeFactoriesTechLevel(int tech_level);
29
30     Teamable* getFlag();
31
32     std::map<int, Factory*>& getFactories();
33
34     bool doesTerritorysOwnerChanged();
35
36     int getId();
37
38     ~Territory();
39 };
40
41
42 #endif //Z_TPGRUPAL_TERRITORY_H

```

jun 27, 17 14:46

## territory.cpp

Page 1/1

```

1  //
2  // Created by rodian on 10/06/17.
3  //
4
5  #include "territory.h"
6  #define FLAGWIDTH 2
7  #define FLAGHEIGHT 3
8  Territory::Territory(const std::map<int, Factory *> &factories, Position flag,
9                      Size size, int id) :
10 Teamable("Neutral",territory_size),factories(factories) ,
11 flag("Neutral",Size(flag.getX(),flag.getY(),FLAGWIDTH,FLAGHEIGHT)),
12 territory_size(size), id(id){}
13
14 void Territory::grabFlag(std::string& new_team) {
15     changed = true;
16     this->flag.changeTeam(new_team);
17     this->changeTeam(new_team);
18     for (auto fac: factories) {
19         fac.second->changeTeam(new_team);
20     }
21 }
22
23 int Territory::getTechLevel() {
24     return (int)factories.size();
25 }
26
27 void Territory::changeFactoriesTechLevel(int tech_level) {
28     for (auto fac: factories) {
29         fac.second->changeTechLevel(tech_level);
30     }
31 }
32
33 std::map<int, Factory*> &Territory::getFactories() {
34     return factories;
35 }
36
37 bool Territory::doesTerritorysOwnerChanged() {
38     if (this->flag.getTeam() != this->team) {
39         std::string new_team = flag.getTeam();
40         grabFlag(new_team);
41     }
42     bool tmp = changed;
43     changed = false;
44     return tmp;
45 }
46
47 int Territory::getId() {
48     return id;
49 }
50
51 Teamable *Territory::getFlag() {
52     return &flag;
53 }
54
55 Territory::~Territory() {
56     for(auto& f: factories) {
57         delete(f.second);
58     }
59     factories.clear();
60 }
61
62

```

jun 27, 17 14:46

## Terrain.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_TERRAIN_H
2  #define Z_TPGRUPAL_TERRAIN_H
3
4  #include <iostream>
5  #include <string>
6
7  // Class Terrain to be set on every cell
8  class Terrain {
9      private:
10         std::string kind;
11         double factor;
12
13     public:
14         // Terrain is of constant kind and factor for moving over it
15         Terrain(std::string& kind, int factor );
16
17         // Returns string to the kind of terrain
18         std::string getKind() const;
19
20         // Returns the moving factor
21         double getFactor() const;
22
23         ~Terrain();
24     };
25
26
27 #endif //Z_TPGRUPAL_TERRAIN_H

```

jun 27, 17 14:46

**Terrain.cpp**

Page 1/1

```

1  //
2  // Created by rodian on 13/05/17.
3  //
4
5  #include "Terrain.h"
6
7  Terrain::Terrain(std::string &kind, int factor) :
8      kind(kind), factor(factor) {}
9
10 std::string Terrain::getKind() const{
11     return this->kind;
12 }
13
14 double Terrain::getFactor() const {
15     return this->factor;
16 }
17
18 Terrain::~Terrain() {}
19
20
21

```

jun 27, 17 14:46

**team.h**

Page 1/1

```

1  //
2  // Created by rodian on 12/06/17.
3  //
4
5  #ifndef Z_TPGRUPAL_TEAM_H
6  #define Z_TPGRUPAL_TEAM_H
7
8
9  #include <vector>
10 #include "playerInfo.h"
11
12 class Team {
13 private:
14     std::vector<PlayerInfo> players;
15     int team_id;
16 public:
17     Team(std::vector<PlayerInfo>& players,int team_id);
18
19     bool doesTeamLose();
20
21     std::vector<PlayerInfo>& getPlayersInfo();
22 };
23
24
25 #endif //Z_TPGRUPAL_TEAM_H

```

jun 27, 17 14:46

team.cpp

Page 1/1

```

1  //
2  // Created by rodian on 12/06/17.
3  //
4
5  #include "team.h"
6
7  Team::Team(std::vector<PlayerInfo>& players, int team_id) :
8      players(players), team_id(team_id) {}
9
10 bool Team::doesTeamLose() {
11     bool lose = true;
12     // if any of the fortress of the team is alive, they didn't lose
13     for (auto& p: players) {
14         if (p.checkIfFortressLives()) {
15             lose = false;
16         }
17         if (!p.areYouStillConected()) {
18             lose = true;
19         }
20     }
21     return lose;
22 }
23
24 std::vector<PlayerInfo> &Team::getPlayersInfo() {
25     return this->players;
26 }

```

jun 27, 17 14:46

teamable.h

Page 1/1

```

1  //
2  // Created by rodian on 22/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_TEAMABLE_H
6  #define Z_TPGRUPAL_TEAMABLE_H
7
8  #include <iostream>
9  #include "size.h"
10
11 class Teamable {
12 protected:
13     std::string team;
14     Size obj_size;
15     bool changed;
16
17 public:
18     Teamable(std::string team, Size obj_size);
19
20     // Builds a Teamable with "Neutral" team
21     Teamable(Size size);
22
23     // Changes the team of the object
24     void changeTeam(std::string team);
25
26     // Returns the actual team of the object
27     std::string getTeam() const;
28
29     // Returns the central position of the object
30     Position getPosition() const;
31
32     // Returns true if is there a collision with the object
33     bool isThereACollision(Size& size);
34
35     Size getSize() const;
36
37     bool haveYouChanged();
38 };
39
40
41 #endif //Z_TPGRUPAL_TEAMABLE_H

```

jun 27, 17 14:46

teamable.cpp

Page 1/1

```

1  //
2  // Created by rodian on 22/05/17.
3  //
4
5  #include "teamable.h"
6
7  Teamable::Teamable(std::string team, Size size) : team(team), obj_size(size),
8                                                    changed(false){}
9
10 Teamable::Teamable(Size size) : team("Neutral"), obj_size(size),
11                                changed(false) {}
12
13 void Teamable::changeTeam(std::string team) {
14     this->changed = true;
15     this->team = team;
16 }
17
18 std::string Teamable::getTeam() const {
19     return this->team;
20 }
21
22 Position Teamable::getPosition() const {
23     return obj_size.getPosition();
24 }
25
26 bool Teamable::isThereACollision(Size& other) {
27     return obj_size.isThereACollision(other);
28 }
29
30 Size Teamable::getSize() const {
31     return this->obj_size;
32 }
33
34 bool Teamable::haveYouChanged() {
35     bool tmp = changed;
36     changed = false;
37     return tmp;
38 }
39

```

jun 27, 17 14:46

size.h

Page 1/1

```

1  //
2  // Created by rodian on 17/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_SIZE_H
6  #define Z_TPGRUPAL_SIZE_H
7
8  #include "position.h"
9
10 //Class Size to represent the space on the map that an Occupant is standing on
11 class Size {
12 private:
13     // (x,y) are the coordinates of the top-left corner of the object
14     // all Occupants on game are Four-sided
15     Position position;
16     int width, height;
17
18 public:
19     // width and lenght must be even numbers
20     Size(int x, int y, int width, int height);
21
22     // Returns the x position on the map
23     Position getPosition() const;
24
25     // Returns the width of the Object of this size
26     int getWidth() const;
27
28     // Returns the lenght of the Object of this size
29     int getHeight() const;
30
31     // Changes the position of the Object of this size to a new (x,y)
32     void moveTo(int x, int y);
33
34     // Returns True is this Size collisions with other.
35     // Meant to check collision between different objects.
36     // To fully be sure there is no collison, use this method on the size of
37     // the other object as well
38     bool isThereACollision(Size& other);
39
40     // If other is partly or completely outside returns true.
41     // If other is all inside this returns false.
42     bool areYouHalfOutSide(Size& other);
43
44     // Returns true if the position (x,y) received is inside this Size
45     bool areYouOnThisPoint(int x, int y);
46
47     ~Size();
48
49     // Writes on max and min the maximum value on 'x' coordinate that this size
50     // has and the minimum value on min
51     void calculateMaxAndMinForX(int& max, int& min);
52
53     // Writes on max and min the maximum value on 'y' coordinate that this size
54     // has and the minimum value on min
55     void calculateMaxAndMinForY(int& max, int& min);
56 };
57
58
59 #endif //Z_TPGRUPAL_SIZE_H

```



jun 27, 17 14:46

size.cpp

Page 1/2

```

1  //
2  // Created by rodian on 17/05/17.
3  //
4
5  #include "size.h"
6
7  Size::Size(int x, int y, int width, int height) : position(x,y),
8                                                    width(width), height(height){}
9
10 Position Size::getPosition() const {
11     return this->position;
12 }
13
14 int Size::getWidth() const {
15     return width;
16 }
17
18 int Size::getHeight() const {
19     return height;
20 }
21
22 bool Size::isThereACollision(Size &other) {
23     int x_max, x_min, y_max, y_min;
24     bool collision = false;
25     this->calculateMaxAndMinForX(x_max, x_min);
26     this->calculateMaxAndMinForY(y_max, y_min);
27
28     if (other.areYouOnThisPoint(x_max, y_max) ∨
29         other.areYouOnThisPoint(x_max, y_min) ∨
30         other.areYouOnThisPoint(x_min, y_max) ∨
31         other.areYouOnThisPoint(x_min, y_min)) {
32         collision = true;
33     }
34
35     other.calculateMaxAndMinForX(x_max, x_min);
36     other.calculateMaxAndMinForY(y_max, y_min);
37     if (this->areYouOnThisPoint(x_max, y_max) ∨
38         this->areYouOnThisPoint(x_max, y_min) ∨
39         this->areYouOnThisPoint(x_min, y_max) ∨
40         this->areYouOnThisPoint(x_min, y_min)) {
41         collision = true;
42     }
43
44     return collision;
45 }
46
47 bool Size::areYouHalfOutSide(Size &other) {
48     int x_max, x_min, y_max, y_min;
49     other.calculateMaxAndMinForX(x_max, x_min);
50     other.calculateMaxAndMinForY(y_max, y_min);
51
52     // If a point is not inside returns true
53     return (¬this->areYouOnThisPoint(x_max, y_max) ∨
54         ¬this->areYouOnThisPoint(x_max, y_min) ∨
55         ¬this->areYouOnThisPoint(x_min, y_max) ∨
56         ¬this->areYouOnThisPoint(x_min, y_min));
57 }
58
59 bool Size::areYouOnThisPoint(int x_pos, int y_pos) {
60     int x_max, x_min, y_max, y_min;
61     this->calculateMaxAndMinForX(x_max, x_min);
62     this->calculateMaxAndMinForY(y_max, y_min);
63
64     return ((x_pos < x_max) ∧ (x_pos ≥ x_min) ∧
65         (y_pos < y_max) ∧ (y_pos ≥ y_min));
66 }

```

jun 27, 17 14:46

size.cpp

Page 2/2

```

67 }
68
69 void Size::calculateMaxAndMinForX(int &max, int &min) {
70     max = position.getX() + width;
71     min = position.getX();
72 }
73
74 void Size::calculateMaxAndMinForY(int &max, int &min) {
75     max = position.getY() + height;
76     min = position.getY();
77 }
78
79 void Size::moveTo(int x, int y) {
80     position.moveTo(x,y);
81 }
82
83 Size::~Size() {}
84

```

jun 27, 17 14:46

server.h

Page 1/1

```

1  //
2  // Created by rodian on 27/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_SERVER_H
6  #define Z_TPGRUPAL_SERVER_H
7
8
9  #include "../common/socket.h"
10 #include "../common/messenger.h"
11 #include "../common/Thread.h"
12 #include "menu.h"
13
14 class Server: public Thread {
15 private:
16     int port;
17     Socket socket;
18     bool running;
19     Menu& menu;
20
21 public:
22     // Recieves the arguments to build the Control Unit
23     explicit Server(unsigned int port, Menu &menu);
24
25     // use to start the process of of accepting clients
26     void run();
27
28     // Shuts down the acceptor Socket and stops run()
29     void stop();
30
31     ~Server();
32 };
33
34 #endif //Z_TPGRUPAL_SERVER_H
35
```

jun 27, 17 14:46

server.cpp

Page 1/1

```

1  //
2  // Created by rodian on 27/05/17.
3  //
4
5  #include "server.h"
6  #include <sstream>
7
8  Server::Server(unsigned int port, Menu &menu) : socket(port),
9                                                  running(true),
10                                                  menu(menu),
11                                                  port(port) {}
12
13 void Server::run() {
14     try {
15         int i = 0;
16         while(this->running) {
17             Socket new_client = this->socket.accept_client();
18             Messenger* messenger = new Messenger(new_client);
19             std::string id_new_player;
20             id_new_player = "Player" + i;
21             bool added = menu.addPlayer(messenger, menu, id_new_player);
22             while (!added) {
23                 std::string new_player;
24                 ++i;
25                 new_player = "Player" + std::to_string(i);
26                 added = menu.addPlayer(messenger, menu, id_new_player);
27             }
28         }
29     } catch (SocketError& e) {
30         std::string error = e.what();
31         std::cerr << error << std::endl;
32     }
33 }
34
35 void Server::stop() {
36     this->running = false;
37     this->socket.shutdown();
38 }
39
40 Server::~~Server() {}
41
```

jun 27, 17 14:46

position.h

Page 1/1

```

1  //
2  // Created by rodian on 18/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_POSITION_H
6  #define Z_TPGRUPAL_POSITION_H
7
8  // Position class to keep the coordinates of the map
9  class Position {
10 private:
11     int x, y;
12 public:
13     Position(int x, int y);
14
15     // Returns the x coordinate
16     int getX() const;
17
18     // Returns the y coordinate
19     int getY() const;
20
21     // Changes the position to the (x,y) received
22     void moveTo(int x, int y);
23 };
24
25
26 #endif //Z_TPGRUPAL_POSITION_H

```

jun 27, 17 14:46

position.cpp

Page 1/1

```

1  //
2  // Created by rodian on 18/05/17.
3  //
4
5  #include "position.h"
6
7  Position::Position(int x, int y) : x(x), y(y) {}
8
9  int Position::getX() const {
10     return this->x;
11 }
12
13 int Position::getY() const {
14     return this->y;
15 }
16
17 void Position::moveTo(int x, int y) {
18     this->x = x;
19     this->y = y;
20 }

```

jun 27, 17 14:46

## playerInfo.h

Page 1/1

```

1  //
2  // Created by rodian on 12/06/17.
3  //
4
5  #ifndef Z_TPGRUPAL_PLAYERINFO_H
6  #define Z_TPGRUPAL_PLAYERINFO_H
7
8
9  #include <messenger.h>
10 #include "factory.h"
11 #include "territory.h"
12
13 class PlayerInfo {
14 private:
15     std::string id;
16     Factory* fortress;
17     int tech_level;
18     Messenger* player_messenger;
19     std::vector<Territory*> territories;
20
21 public:
22     PlayerInfo(std::string id ,Factory* fortress);
23
24     PlayerInfo(std::string id);
25
26     std::string getPlayerId() const;
27
28     bool checkIfFortressLives();
29
30     int getTechLevel();
31
32     void increaseTechLevel();
33
34     void decreaseTechLevel();
35
36     Factory* getFortress();
37
38     void addMessenger(Messenger* player_messenger);
39
40     Messenger* getMessenger();
41
42     void addTerritory(Territory* territory);
43
44     void recalculateTechLevel();
45
46     void eliminateThisTerritory(Territory* territory);
47
48     bool areYouStillConected();
49
50     void addFortress(Factory* fortress);
51
52     ~PlayerInfo();
53 };
54
55 #endif //Z_TPGRUPAL_PLAYERINFO_H

```

jun 27, 17 14:46

## playerInfo.cpp

Page 1/2

```

1  //
2  // Created by rodian on 12/06/17.
3  //
4
5  #include "playerInfo.h"
6
7  PlayerInfo::PlayerInfo(std::string id, Factory *fortress) :
8      id(id), fortress(fortress), tech_level(1) {}
9
10
11 PlayerInfo::PlayerInfo(std::string id) : id(id), tech_level(0) {}
12
13 std::string PlayerInfo::getPlayerId() const {
14     return id;
15 }
16
17 bool PlayerInfo::checkIfFortressLives() {
18     return fortress->areYouAlive();
19 }
20
21 int PlayerInfo::getTechLevel() {
22     return tech_level;
23 }
24
25 void PlayerInfo::increaseTechLevel() {
26     this->tech_level += 1;
27 }
28
29 void PlayerInfo::decreaseTechLevel() {
30     this->tech_level -= 1;
31 }
32
33 Factory *PlayerInfo::getFortress() {
34     return fortress;
35 }
36
37 void PlayerInfo::addMessenger(Messenger *messenger) {
38     this->player_messenger = messenger;
39 }
40
41 Messenger *PlayerInfo::getMessenger() {
42     return this->player_messenger;
43 }
44
45 void PlayerInfo::addTerritory(Territory *territory) {
46     territories.push_back(territory);
47     recalculateTechLevel();
48 }
49
50 void PlayerInfo::recalculateTechLevel() {
51     tech_level = 0;
52     for (auto& t: territories) {
53         tech_level += t->getTechLevel();
54     }
55
56     for (auto& t: territories) {
57         t->changeFactoriesTechLevel(tech_level);
58     }
59 }
60
61 void PlayerInfo::eliminateThisTerritory(Territory* territory) {
62     std::vector<Territory*>::iterator it = territories.begin();
63     for (; it != territories.end(); ) {
64         if (territory->getId() == (*it)->getId()) {
65             it = territories.erase(it);
66         } else {

```

jun 27, 17 14:46

playerInfo.cpp

Page 2/2

```

67         ++it;
68     }
69 }
70     recalculateTechLevel();
71 }
72
73 PlayerInfo::~PlayerInfo() {
74     player_messenger = nullptr;
75 }
76
77 bool PlayerInfo::areYouStillConected() {
78     return player_messenger->isConnected();
79 }
80
81 void PlayerInfo::addFortress(Factory *fortress) {
82     this->fortress = fortress;
83 }
84

```

jun 27, 17 14:46

player.h

Page 1/2

```

1  //
2  // Created by rodian on 29/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_PLAYER_H
6  #define Z_TPGRUPAL_PLAYER_H
7
8  #include <iostream>
9  #include "../common/messenger.h"
10 #include "../common/Thread.h"
11 #include "menu.h"
12 #include "lobby.h"
13
14 class CommandMonitor;
15 class ControlUnit;
16 class Lobby;
17 class Menu;
18 class Game;
19
20 class Player: public Thread {
21 private:
22     Messenger* messenger;
23     std::string id;
24     int color;
25     bool conected,on_menu,on_lobby ,playing,ready;
26     CommandMonitor* commands;
27     ControlUnit* control;
28     Lobby* lobby;
29     Menu& menu;
30
31 public:
32     Player(Messenger *messenger, Menu &menu, std::string& id);
33
34     void run();
35
36     void updateInfo(std::string& info);
37
38     void addLobby(Lobby* lobby);
39
40     void addControlUnit(ControlUnit* control, CommandMonitor* commands);
41
42     Messenger* getMessenger();
43
44     void shutDown();
45
46     std::string getId() const;
47
48     void getInGame();
49
50     bool areYouReady();
51
52     void resetReady();
53
54     bool areYouInLobby();
55
56     ~Player();
57
58 private:
59     void processMenuCommands(std::string& cmd);
60
61     void processLobbyCommands(std::string& cmd);
62
63     std::string getNextData(std::string& line);
64 };
65
66

```

jun 27, 17 14:46

player.h

Page 2/2

```
67 #endif //Z_TPGRUPAL_PLAYER_H
```

jun 27, 17 14:46

player.cpp

Page 1/3

```
1 //
2 // Created by rodian on 29/05/17.
3 //
4
5 #include <sstream>
6 #include "player.h"
7
8 #define RETURNMENU "returnmenu"
9 #define OK "ok"
10
11 Player::Player(Messenger *msg, Menu &menu, std::string& id) :
12     messenger(msg), id(id), conected(true), on_menu(true), on_lobby(false),
13     playing(false), menu(menu), ready(false) {}
14
15 void Player::run() {
16     try {
17         while (messenger->isConnected()) {
18             std::string new_cmd = messenger->recieveMessage();
19
20             std::cerr << "Player " << id << " ejecuta " << new_cmd << std::endl;
21             if (on_menu) {
22                 processMenuCommands(new_cmd);
23             } else if (on_lobby) {
24                 processLobbyCommands(new_cmd);
25             } else if (new_cmd == RETURNMENU) {
26                 this->playing = false;
27                 this->on_menu = true;
28                 messenger->sendMessage(this->menu.getLobbiesInfo());
29             } else if (playing) {
30                 commands->addCommand(this->id, new_cmd, control);
31             }
32         }
33     } catch (SocketError e) {
34         conected = false;
35         if (on_lobby || playing) {
36             lobby->disconnectPlayer(this);
37         }
38     }
39 }
40
41 void Player::updateInfo(std::string &info) {
42     messenger->sendMessage(info);
43 }
44
45 void Player::addControlUnit(ControlUnit *control, CommandMonitor* commands) {
46     this->playing = true;
47     this->control = control;
48     this->commands = commands;
49 }
50
51 Messenger *Player::getMessenger() {
52     return messenger;
53 }
54
55 void Player::addLobby(Lobby* lobby) {
56     this->lobby = lobby;
57     on_lobby = true;
58 }
59
60 void Player::processMenuCommands(std::string &full_cmd) {
61     std::string cmd = getNextData(full_cmd);
62     if (cmd == "createlobby") {
63         this->menu.createNewLobby(this);
64         on_menu = false;
65     } else if (cmd == "joinlobby") {
66         std::string lobby_id = getNextData(full_cmd);
```

jun 27, 17 14:46

player.cpp

Page 2/3

```

67     int id = std::stoi(lobby_id);
68     if (this->menu.addToLobby(id, this))
69         on_menu = false;
70 } else if (cmd == "lobbyinfo") {
71     messenger->sendMessage(this->menu.getLobbiesInfo());
72 } else if (cmd == "changenname") {
73     std::string new_name = getNextData(full_cmd);
74     std::string ans = this->menu.changeName(new_name);
75     if (ans == "ok")
76         this->id = new_name;
77     messenger->sendMessage(ans);
78 } else {
79     messenger->sendMessage("Invalid cmd");
80 }
81 }
82
83 void Player::processLobbyCommands(std::string &full_cmd) {
84     std::string cmd = getNextData(full_cmd);
85     if (cmd == "startgame") {
86         std::string map = getNextData(full_cmd);
87         this->lobby->startGame(map);
88     } else if (cmd == "ready") {
89         this->ready = true;
90         this->lobby->ready();
91     } else if (cmd == "unready") {
92         this->ready = false;
93         this->lobby->unReady();
94     } else if (cmd == "exitlobby") {
95         this->ready = false;
96         this->lobby->unReady();
97         this->on_lobby = false;
98         this->on_menu = true;
99         this->lobby->exitLobby(this);
100    } else if (cmd == "mapsinfo") {
101        messenger->sendMessage(lobby->get_loaded_maps());
102    } else {
103        messenger->sendMessage("Invalid cmd");
104    }
105 }
106
107 void Player::shutDown() {
108     conected = false;
109     messenger->shutdown();
110 }
111
112 std::string Player::getId() const {
113     return id;
114 }
115
116 std::string Player::getNextData(std::string& line) {
117     std::size_t found = line.find('-');
118     std::string data = line.substr(0, found);
119     line.erase(0, found+1);
120     return data;
121 }
122
123 void Player::getInGame() {
124     // Notify the client the game is starting
125     std::stringstream msg;
126     msg << "startgame-" << OK;
127     messenger->sendMessage(msg.str());
128     this->on_lobby = false;
129     this->playing = true;
130 }
131
132 bool Player::areYouReady() {

```

jun 27, 17 14:46

player.cpp

Page 3/3

```

133     return this->ready;
134 }
135
136 Player::~Player() {
137     commands = nullptr;
138     control = nullptr;
139     lobby = nullptr;
140 }
141
142 void Player::resetReady() {
143     this->ready = false;
144 }
145
146 bool Player::areYouInLobby() {
147     return on_lobby;
148 }

```

jun 27, 17 14:46	Occupant.h	Page 1/1
1	<b>#ifndef</b> Z_TPGRUPAL_OCCUPANT_H	
2	<b>#define</b> Z_TPGRUPAL_OCCUPANT_H	
3		
4	<b>#include</b> <iostream>	
5	<b>#include</b> "size.h"	
6	<b>#include</b> "teamable.h"	
7		
8		
9	<i>// Class Occupant so any object knows where is on the map</i>	
10	class Occupant: public Teamable {	
11	protected:	
12	int id, life_points, damage_recv;	
13	<i>// Size occ_size;</i>	
14	std::string type;	
15	bool disappear;	
16		
17	public:	
18	<i>// Constructor for Occupant on a specific position saved in Size and it's id</i>	
19	Occupant(int id, int life, std::string type, Size occ_size);	
20		
21	<i>// Meant to create a null Occupant</i>	
22	Occupant();	
23		
24	<i>// Returns the id of the object</i>	
25	std::string getType() <b>const</b> ;	
26		
27	void reduceLifeBy(int dmg);	
28		
29	bool areYouAlive();	
30		
31	<i>// bool isThereACollision(Size&amp; size);</i>	
32		
33	int getLifeLeft();	
34		
35	int getId();	
36		
37	void mustDisappear();	
38		
39	bool doYouNeedToDisappear();	
40		
41	~Occupant();	
42	};	
43		
44		
45	<b>#endif</b> <i>//Z_TPGRUPAL_OCCUPANT_H</i>	

jun 27, 17 14:46	Occupant.cpp	Page 1/1
1	<i>//</i>	
2	<i>// Created by rodian on 13/05/17.</i>	
3	<i>//</i>	
4		
5	<b>#include</b> "Occupant.h"	
6		
7	Occupant::Occupant(int id, int life, std::string type, Size position) :	
8	Teamable(position), id(id), life_points(life), type(type)/*,	
9	occ_size(position)*/, damage_recv(0), disappear(false) {}	
10		
11	Occupant::Occupant() :	
12	Teamable(Size(3,3,3)), id(-1), life_points(0), type("nullOccupant")/*,	
13	occ_size(position)*/, damage_recv(0) {}	
14		
15	std::string Occupant::getType() <b>const</b> {	
16	<b>return</b> this->type;	
17	}	
18		
19	void Occupant::reduceLifeBy(int dmg) {	
20	this->damage_recv += dmg;	
21	this->changed = true;	
22	}	
23		
24	bool Occupant::areYouAlive() {	
25	<b>return</b> ((life_points - damage_recv) > 0);	
26	}	
27		
28	<i>//bool Occupant::isThereACollision(Size &amp;other) {</i>	
29	<i>// return occ_size.isThereACollision(other);</i>	
30	<i>//}</i>	
31		
32	int Occupant::getLifeLeft() {	
33	<b>if</b> (life_points - damage_recv > 0) {	
34	<b>return</b> life_points - damage_recv;	
35	} <b>else</b> {	
36	<b>return</b> 0;	
37	}	
38	}	
39		
40	Occupant::~~Occupant() {}	
41		
42	int Occupant::getId() {	
43	<b>return</b> id;	
44	}	
45		
46	bool Occupant::doYouNeedToDisappear() {	
47	<b>return</b> disappear;	
48	}	
49		
50	void Occupant::mustDisappear() {	
51	disappear = true;	
52	}	
53		
54		



jun 27, 17 14:46

node.h

Page 1/1

```

1  //
2  // Created by rodian on 15/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_NODE_H
6  #define Z_TPGRUPAL_NODE_H
7
8  #include "cell.h"
9  #include "size.h"
10 #include <iostream>
11 #include <vector>
12
13 // Class Node meant to be use in the calculation of A* algorithm
14 class Node {
15 private:
16     int h_value, g_value;
17     Node* parent;
18     bool was_visited;
19     Size size;
20
21 public:
22     Node(int x, int y, int width, int lenght);
23
24     void setHValue(int h);
25
26     void setGValue(int g, int terrain_factor);
27
28     int getGValue();
29
30     int getFValue() const;
31
32     int getHValue() const;
33
34     void setNewParent(Node* parent);
35
36     Node* getParent() const;
37
38     Position getPosition() const;
39
40     Size getSize() const;
41
42     bool beenSeen() const;
43
44     int getFValueIfGWere(int g, int terrain_factor);
45
46     void clean();
47
48     ~Node();
49 };
50
51 #endif //Z_TPGRUPAL_NODE_H

```

jun 27, 17 14:46

node.cpp

Page 1/1

```

1  //
2  // Created by rodian on 15/05/17.
3  //
4
5  #include "node.h"
6
7  Node::Node(int x, int y, int width, int lenght) : h_value(0), g_value(0),
8                                                    was_visited(false), size(x, y, width, lenght) {}
9
10 void Node::setHValue(int h) {
11     this->h_value = h;
12 }
13
14 void Node::setGValue(int g, int terrain_factor) {
15     this->g_value = (g + 4)*terrain_factor*terrain_factor;
16     this->was_visited = true;
17 }
18
19 int Node::getGValue() {
20     return this->g_value;
21 }
22
23 int Node::getFValue() const {
24     return (h_value + g_value);
25 }
26
27 int Node::getHValue() const {
28     return this->h_value;
29 }
30
31 void Node::setNewParent(Node *parent) {
32     this->parent = parent;
33 }
34
35 Node* Node::getParent() const {
36     return this->parent;
37 }
38
39 Position Node::getPosition() const {
40     return size.getPosition();
41 }
42
43 Size Node::getSize() const {
44     return this->size;
45 }
46
47 bool Node::beenSeen() const {
48     return this->was_visited;
49 }
50
51 int Node::getFValueIfGWere(int g, int terrain_factor) {
52     return (h_value + (g + 4)*terrain_factor);
53 }
54
55 void Node::clean() {
56     this->h_value = 0;
57     this->g_value = 0;
58     this->was_visited = false;
59 }
60
61 Node::~Node() {
62     this->parent = nullptr;
63 }
64

```

jun 27, 17 14:46

menu.h

Page 1/1

```

1 //
2 // Created by rodian on 29/05/17.
3 //
4
5 #ifndef Z_TPGRUPAL_MENU_H
6 #define Z_TPGRUPAL_MENU_H
7
8
9 #include <vector>
10 #include <iostream>
11 #include <mutex>
12 #include "player.h"
13 class Player;
14 class Lobby;
15
16 class Menu {
17 private:
18     std::vector<Player*> players;
19     std::mutex m;
20     int lobby_counter;
21     std::vector<Lobby*> lobbies;
22     std::string& config;
23 public:
24     Menu(std::string& config);
25
26     bool addPlayer(Messenger* msgr, Menu& menu, std::string player_id);
27
28     void createNewLobby(Player* player);
29
30     std::string getLobbiesInfo();
31
32     std::string changeName(std::string &new_name);
33
34     bool addToLobby(int id_lobby, Player *player);
35
36     void shutDown();
37
38     void disconnectPlayer( Player *player);
39
40     ~Menu();
41 };
42
43
44 #endif //Z_TPGRUPAL_MENU_H

```

jun 27, 17 14:46

menu.cpp

Page 1/3

```

1 //
2 // Created by rodian on 29/05/17.
3 //
4
5 #include <sstream>
6 #include <string>
7 #include "menu.h"
8
9 #define ERROR_MSG "joinlobby-error"
10 #define OK_MSG "joinlobby-ok"
11
12 Menu::Menu(std::string& config) : lobby_counter(0), config(config) {}
13
14 bool Menu::addPlayer(Messenger *msgr, Menu& menu, std::string player_id) {
15     Lock l(m);
16     // check desconected players
17     std::vector<Player *>::iterator p = players.begin();
18     for (; p != players.end(); ) {
19         if (!(*p)→getMessenger()→isConnected()) {
20             (*p)→shutDown();
21             (*p)→join();
22             delete((*p));
23             p = players.erase(p);
24         } else {
25             ++p;
26         }
27     }
28
29     for(Player* p : players) {
30         if (p→getId() == player_id) {
31             return false;
32         }
33     }
34     this→players.push_back(new Player(msgr, menu, player_id));
35     this→players.back()→start();
36     return true;
37 }
38
39 void Menu::createNewLobby(Player* player) {
40     Lock l(m);
41     Lobby* new_lobby = new Lobby(lobby_counter++, config,m);
42     lobbies.emplace_back(new_lobby);
43     lobbies.back()→addPlayer(player);
44     player→addLobby(new_lobby);
45     player→getMessenger()→sendMessage(OK_MSG);
46 }
47
48 std::string Menu::getLobbiesInfo() {
49     Lock l(m);
50     std::string info = "lobbyinfo-";
51     std::vector<Lobby *>::iterator it = lobbies.begin();
52     for (; it != lobbies.end(); ) {
53         if (!(*it)→haveGameFinished()) {
54             info += std::to_string((*it)→get_id()) + "-";
55             ++it;
56         } else {
57             (*it)→shutDown();
58             delete(*it);
59             it = lobbies.erase(it);
60             --lobby_counter;
61             // check desconected players
62             std::vector<Player *>::iterator p = players.begin();
63             for (; p != players.end(); ) {
64                 if (!(*p)→getMessenger()→isConnected()) {
65                     (*p)→shutDown();
66                     (*p)→join();

```

jun 27, 17 14:46

menu.cpp

Page 2/3

```

67         delete((*p));
68         p = players.erase(p);
69     } else {
70         ++p;
71     }
72 }
73 }
74 }
75 return info;
76 }
77
78 bool Menu::addToLobby(int id_lobby, Player* player) {
79     Lock l(m);
80     for (Lobby* lobby : lobbies) {
81         if (lobby->get_id() == id_lobby) {
82             if (lobby->addPlayer(player)) {
83                 player->addLobby(lobby);
84                 player->getMessenger()->sendMessage(OK_MSG);
85                 return true;
86             } else {
87                 player->getMessenger()->sendMessage(ERROR_MSG);
88                 return false;
89             }
90         }
91     }
92     player->getMessenger()->sendMessage(ERROR_MSG);
93     return false;
94 }
95
96 void Menu::shutDown() {
97     for(auto p: players) {
98         p->shutDown();
99         p->join();
100        delete(p);
101    }
102
103    for(auto l: lobbies) {
104        l->shutDown();
105        delete(l);
106    }
107 }
108
109 Menu::~Menu() {}
110
111 std::string Menu::changeName(std::string &new_name) {
112     Lock l(m);
113     for(Player* p : players) {
114         if (p->getId() == new_name) {
115             return "error";
116         }
117     }
118     return "ok";
119 }
120
121 void Menu::disconnectPlayer(Player *player) {
122     Lock l(m);
123     std::vector<Player*>::iterator it = players.begin();
124     for(; it != players.end(); ++it) {
125         if ((*it)->getId() == player->getId()) {
126             (*it)->shutDown();
127             (*it)->join();
128             delete((*it));
129             players.erase(it);
130             break;
131         }
132     }

```

jun 27, 17 14:46

menu.cpp

Page 3/3

```

133 }
134

```

```

jun 27, 17 14:46      MapLoader.h      Page 1/1
1  #ifndef Z_TPGRUPAL_MAPLOADER_H
2  #define Z_TPGRUPAL_MAPLOADER_H
3
4
5  #include <vector>
6  #include <pugixml.hpp>
7  #include <memory>
8  #include "map.h"
9  #include "Occupant.h"
10 #include "cell.h"
11 #include "unit.h"
12 #include "factory.h"
13 #include "territory.h"
14
15 class MapLoader {
16     std::vector<std::vector<Cell>> map;
17     std::vector<Occupant*> occupants;
18     std::vector<Unit> units;
19     std::string map_string;
20     std::string &config;
21     std::vector<UnitMold*> unit_mold;
22     std::vector<UnitMold*> vehicle_mold;
23     std::map<std::string, Weapon> weapons;
24     std::shared_ptr<Map> game_map;
25     std::vector<Factory*> forts;
26
27     std::vector<Territory*> territories;
28     int internal_positions;
29 public:
30     MapLoader(std::string path, std::string& config);
31
32     ~MapLoader();
33
34     std::vector<Occupant*> getOccupants();
35
36     std::vector<Unit> getUnits();
37
38     std::shared_ptr<Map> get_map();
39
40     std::vector<Factory*> get_forts();
41
42     std::vector<Territory*> get_territories();
43
44 private:
45     void load_structs(const pugi::xml_node &root, const pugi::xml_node &cfg);
46
47     Factory* create_factory(int id, int hp, std::string& type, Size size);
48
49     void load_unit_molds(pugi::xml_node node);
50     void load_weapons(pugi::xml_node weapons);
51     void create_map();
52     void load_territories(const pugi::xml_node &structs_cfg,
53                          const pugi::xml_node &root, int id_counter);
54
55     void create_territory(int hp, const pugi::xml_node &territory,
56                          int &id_counter,
57                          std::map<int, Factory*> &factories_in_territory);
58
59     void load_vehicle_molds(const pugi::xml_node &vehicles);
60
61     void load_mold(std::vector<UnitMold*> &mold,
62                  const pugi::xml_node& source);
63 };
64
65
66 #endif //Z_TPGRUPAL_MAPLOADER_H

```

```

jun 27, 17 14:46      MapLoader.cpp      Page 1/4
1  #include <map>
2  #include "MapLoader.h"
3  #include <pugixml.hpp>
4  #include <sstream>
5
6
7  const std::map<std::string, int> terrain_factor {
8      {std::string("Tierra"), int(1)},
9      {std::string("Agua"), int(7/10)},
10     {std::string("Carretera"), int(15/10)},
11     {std::string("Lava"), int(1000)}
12 };
13
14 MapLoader::MapLoader(std::string path, std::string& config) : config(config) {
15     pugi::xml_document doc;
16     pugi::xml_parse_result result = doc.load_file(path.c_str());
17     if (!result) {
18         std::cout << "ERROR LOADING MAP: " << path << " " <<
19             result.description() << std::endl;
20     }
21     std::stringstream stream;
22     doc.save(stream);
23     map_string = stream.str();
24     // Get root node
25     pugi::xml_node root = doc.child("Map");
26     pugi::xml_node map_node = root.child("Terrain");
27
28     pugi::xml_document cfg;
29     cfg.load_file(config.c_str());
30     pugi::xml_node cfg_node = cfg.child("Config");
31
32     internal_positions = std::stoi(cfg_node.child("Cells").
33         attribute("internal_positions").value());
34
35     // Iterate over every row
36     unsigned int coord_y = 0;
37     auto row = map_node.children().begin();
38     for (; row != map_node.children().end(); ++row) {
39         unsigned int coord_x = 0;
40         // Iterate over every row creating cells
41         auto cell = row->children().begin();
42         for (; cell != row->children().end(); ++cell) {
43             if (map.size() <= (coord_x)) {
44                 map.push_back(std::vector<Cell>());
45             }
46
47             std::string terrain = cell->attribute("terrain").value();
48             std::string structure = cell->attribute("struct").value();
49
50             int factor = terrain_factor.find(terrain)->second;
51
52             // Create a new cell and push it to the row
53             map.at(coord_x).emplace_back(coord_x * internal_positions, coord_y,
54                 internal_positions, internal_positions,
55                 terrain, factor);
56
57             coord_x++;
58         }
59         // Push the whole row to the map
60         coord_y += internal_positions;
61     }
62
63     create_map();
64     load_unit_molds(cfg_node.child("Units"));
65     load_vehicle_molds(cfg_node.child("Vehicles"));
66 }

```

jun 27, 17 14:46

## MapLoader.cpp

Page 2/4

```

67     load_weapons(cfg_node.child("Weapons"));
68     load_structs(root, cfg_node.child("Structs"));
69 }
70
71 void MapLoader::load_structs(const pugi::xml_node &root,
72                             const pugi::xml_node &cfg) {
73
74     int id_counter = 0;
75     pugi::xml_node structs = root.child("Structures");
76     pugi::xml_node structure_cfg = cfg.find_child_by_attribute("type", "Rock");
77     int size_x = std::stoi(structure_cfg.attribute("size_x").value());
78     int size_y = std::stoi(structure_cfg.attribute("size_y").value());
79     int hp = std::stoi(structure_cfg.attribute("hp").value());
80
81     std::string type = structure_cfg.attribute("type").value();
82     for(auto& rock : structs) {
83         int x = std::stoi(rock.attribute("x").value()) *
84             internal_positions;
85         int y = std::stoi(rock.attribute("y").value()) *
86             internal_positions;
87         Occupant* f = new Occupant(id_counter++, hp, type,
88                                     Size(x, y, size_x, size_y));
89         occupants.push_back(f);
90     }
91     load_territories(cfg, root.child("Territories"), id_counter);
92
93 }
94
95 void MapLoader::create_map() {
96     int width = (int) map.at(0).size() * internal_positions;
97     int height = (int) map.size() * internal_positions;
98     int x = 0;
99     int y = 0;
100     game_map = std::shared_ptr<Map>(new Map(x, y, width, height, map,
101                                             &occupants, map_string));
102 }
103
104 MapLoader::~MapLoader() {
105 }
106
107 std::vector<Occupant*> MapLoader::getOccupants() {
108     return this->occupants;
109 }
110
111 std::vector<Unit> MapLoader::getUnits() {
112     return this->units;
113 }
114
115 Factory *MapLoader::create_factory(int id, int hp, std::string &type, Size size)
116 {
117     std::vector<UnitMold*> both = unit_mold;
118     for (UnitMold* vehicle : vehicle_mold) {
119         both.push_back(vehicle);
120     }
121     return (new Factory(id, hp, type, size, both, game_map, weapons));
122 }
123
124 void MapLoader::load_unit_molds(pugi::xml_node units) {
125     load_mold(unit_mold, units);
126 }
127
128 void MapLoader::load_weapons(pugi::xml_node weapons) {
129     for(auto& unit : weapons.children()) {
130         std::string type = unit.attribute("type").value();
131         int width = std::stoi(unit.attribute("size_x").value());

```

jun 27, 17 14:46

## MapLoader.cpp

Page 3/4

```

132     int height = std::stoi(unit.attribute("size_y").value());
133     int speed = std::stoi(unit.attribute("speed").value());
134     int damage = std::stoi(unit.attribute("damage").value());
135     std::string exp = unit.attribute("explosive").value();
136     bool explosive = false;
137     if (exp == "yes")
138         explosive = true;
139
140     Size s(0, 0, width, height);
141     this->weapons.emplace(type, Weapon(type, damage, speed, explosive, s));
142 }
143
144 std::shared_ptr<Map> MapLoader::get_map() {
145     return game_map;
146 }
147
148 void MapLoader::load_territories(const pugi::xml_node &structs_cfg,
149                                 const pugi::xml_node &root, int id_counter) {
150     pugi::xml_node factory_cfg = structs_cfg.
151         find_child_by_attribute("type", "Factory");
152
153     int size_x = std::stoi(factory_cfg.attribute("size_x").value());
154     int size_y = std::stoi(factory_cfg.attribute("size_y").value());
155     int hp = std::stoi(factory_cfg.attribute("hp").value());
156     std::string type = factory_cfg.attribute("type").value();
157
158     for(auto& territory : root.children()) {
159         std::map<int, Factory*> factories_in_territory;
160         for(auto& factory : territory.children()) {
161             std::vector<UnitMold*> mold = unit_mold;
162             std::string name = factory.name();
163             if (name == "VehicleFactory") {
164                 mold = vehicle_mold;
165             }
166
167             int x = std::stoi(factory.attribute("x").value()) *
168                 internal_positions;
169             int y = std::stoi(factory.attribute("y").value()) *
170                 internal_positions;
171             Size s(x, y, size_x, size_y);
172             Factory* f = new Factory(id_counter, hp, type, s, mold, game_map, we
173 apons);
174             factories_in_territory[id_counter] = f;
175             ++id_counter;
176         }
177         create_territory(hp, territory, id_counter, factories_in_territory);
178     }
179 }
180
181 void MapLoader::create_territory(int hp, const pugi::xml_node &territory,
182                                 int &id_counter,
183                                 std::map<int, Factory *> &factories_in_territor
184 y) {
185     std::string name = territory.name();
186     int x = std::stoi(territory.attribute("center_x").value()) *
187         internal_positions;
188     int y = std::stoi(territory.attribute("center_y").value()) *
189         internal_positions;
190     /* If it's a fort we also create a factory */
191     if (name == "Fort") {
192         Factory* f = create_factory(id_counter, hp, name,
193                                     Size(x, y, 20, 20));
194         factories_in_territory[id_counter] = f;
195         forts.push_back(f);
196         ++id_counter;

```

jun 27, 17 14:46

## MapLoader.cpp

Page 4/4

```

196     }
197
198     int min_x = std::stoi(territory.attribute("min_x").value());
199     int min_y = std::stoi(territory.attribute("min_y").value());
200     int max_x = std::stoi(territory.attribute("max_x").value());
201     int max_y = std::stoi(territory.attribute("max_y").value());
202     int width = (min_x + max_x) / 2;
203     int height = (min_y + max_y) / 2;
204     Size flag(x, y, width, height);
205     Position flag_position(x, y);
206     Territory* t = new Territory(factories_in_territory, flag_position, flag,
207 ++id_counter);
208     territories.emplace_back(t);
209 }
210
211 std::vector<Factory *> MapLoader::get_forts() {
212     return forts;
213 }
214
215 std::vector<Territory *> MapLoader::get_territories() {
216     return territories;
217 }
218
219 void MapLoader::load_vehicle_molds(const pugi::xml_node &vehicles) {
220     load_mold(vehicle_mold, vehicles);
221 }
222
223 void MapLoader::load_mold(std::vector<UnitMold *>& mold,
224     const pugi::xml_node &source) {
225     for (auto& unit : source.children()) {
226         std::string type = unit.attribute("type").value();
227         std::string weapon = unit.attribute("weapon").value();
228         int width = std::stoi(unit.attribute("size_x").value());
229         int height = std::stoi(unit.attribute("size_y").value());
230         int hp = std::stoi(unit.attribute("hp").value());
231         int fire_rate = std::stoi(unit.attribute("fire_rate").value());
232         int range = std::stoi(unit.attribute("range").value());
233         int speed = std::stoi(unit.attribute("speed").value());
234         int time = std::stoi(unit.attribute("time").value());
235         int quantity = std::stoi(unit.attribute("quantity").value());
236         int tech_level = std::stoi(unit.attribute("tech_level").value());
237         mold.push_back(
238             new UnitMold(tech_level, hp, range, width, height, speed,
239                 fire_rate, time, quantity, type, weapon));
240     }
241 }
242
243
244

```

jun 27, 17 14:46

## map.h

Page 1/2

```

1 //
2 // Created by rodian on 18/05/17.
3 //
4
5 #ifndef Z_TPGRUPAL_MAP_H
6 #define Z_TPGRUPAL_MAP_H
7
8
9 #include <vector>
10 #include <map>
11 #include "cell.h"
12 #include "Occupant.h"
13 // later written
14 //class Compass;
15 class Unit;
16
17 class Map {
18 private:
19     std::vector<std::vector<Cell>> terrain_map;
20     Size map_size;
21     std::vector<Occupant*>* all_occupants;
22     std::string xml;
23     std::map<std::string, std::string> types;
24     /*std::vector<Unit>& all_units;*/
25
26 public:
27     // Map receives the center position (x,y) and dimensions width and height
28     Map(int x, int y, int width, int height,
29         std::vector<std::vector<Cell>>& terrain_map,
30         std::vector<Occupant*>* all_occupants,
31         std::string& xml);
32
33     // Recieves the coordinates (x,y) and returns the terrain factor on that
34     // position on the map.
35     double getTerrainFactorOn(int x, int y);
36
37     // Returns the name of the type of Terrain
38     std::string getTerrainType(int x, int y);
39
40     // Returns true if the points are empty
41     bool areThisPointsEmpty(Size &size, int id);
42
43     // Returns true if points are empty or it is the Occupant
44     bool areThisPointsEmpty(Size& size, Occupant &shooter, Occupant& occupant);
45
46     // Recieves the size of an object on the position that wants to be walk
47     // Returns true if the object fits and can step to that position
48     bool canIWalkToThisPosition(Size &size, int id);
49
50     // Recieves the size of an object on the position that wants to be walk
51     // Returns true if the object fits and can step to that position ignoring
52     // the occupant from parameter
53     bool canBulletWalkToThisPosition(Size& size, Occupant &shooter,
54         Occupant& target);
55
56     // Returns the width of the map
57     int getWidth();
58
59     // Returns the Heigth of the map
60     int getHeigth();
61
62     bool doesThisPositionExist(int x, int y);
63
64     bool isThereLava(Size& other_size);
65
66     bool thereIsABridge(Size& other_size);

```

jun 27, 17 14:46

map.h

Page 2/2

```

67     std::string& get_map();
68
69     std::vector<Occupant*>& getOccupants();
70
71     void updateOccupants(std::vector<Occupant*>* all_occupants);
72
73     Occupant* checkForEnemiesOn(Size& range, Occupant& unit);
74
75     bool tellIfItIsGrabbable(std::string& type);
76
77     bool tellIfItIsBuilding(std::string& type);
78
79 private:
80     void buildTypeMap();
81 };
82
83
84
85 #endif //Z_TPGRUPAL_MAP_H

```

jun 27, 17 14:46

map.cpp

Page 1/4

```

1  //
2  // Created by rodian on 18/05/17.
3  //
4
5  #include "map.h"
6  #define BUILDING "Building"
7  #define NATURE "Nature"
8  #define UNIT "Unit"
9  #define VEHICLE "Vehicule"
10 #define FLAG "flag"
11
12 Map::Map(int x, int y, int width, int height,
13         std::vector<std::vector<Cell>>& terrain_map,
14         std::vector<Occupant*>* occupants,
15         std::string& xml) : map_size(x,y,width,height),
16         terrain_map(terrain_map), all_occupants(occupants), xml(xml) {
17     this->buildTypeMap();
18 }
19
20 double Map::getTerrainFactorOn(int x, int y) {
21     int w_cell = terrain_map[0][0].getWidthOfCell();
22     int x_pos = x / w_cell;
23     int y_pos = y / w_cell;
24
25     return terrain_map[x_pos][y_pos].getMovementFactor();
26 }
27
28 std::string Map::getTerrainType(int x, int y) {
29     int w_cell = terrain_map[0][0].getWidthOfCell();
30     int x_pos = x / w_cell;
31     int y_pos = y / w_cell;
32
33     return terrain_map[x_pos][y_pos].getTerrainType();
34 }
35
36 bool Map::areThisPointsEmpty(Size &size, int id) {
37     bool no_collision = true;
38     for(auto x: *all_occupants) {
39         if(x->getId() != id
40            ^ x->isThereACollision(size) ^ x->getType() != "Bridge") {
41             no_collision = false;
42             break;
43         }
44     }
45     return no_collision;
46 }
47
48 bool Map::areThisPointsEmpty(Size &size, Occupant &shooter, Occupant &occupant)
49 {
50     bool no_collision = true;
51     for(auto x: *all_occupants) {
52         if(x->getId() != occupant.getId() ^ x->isThereACollision(size)
53            ^ x->getType() != "Bridge" ^ x->getId() != shooter.getId()) {
54             no_collision = false;
55             break;
56         }
57     }
58     return no_collision;
59 }
60
61 int Map::getWidth() {
62     return map_size.getWidth();
63 }
64
65 int Map::getHeigth() {
66     return map_size.getHeight();
67 }

```

jun 27, 17 14:46

map.cpp

Page 2/4

```

66 }
67
68 bool Map::canIWalkToThisPosition(Size &other_size, int id) {
69     bool you_can = true;
70
71     // if the object is stepping out of the map
72     if (map_size.areYouHalfOutSide(other_size))
73         you_can = false;
74     // if the object is stepping into lava
75     if (isThereLava(other_size)) {
76         you_can = false;
77         if (thereIsABridge(other_size))
78             you_can = true;
79     }
80     if (!areThisPointsEmpty(other_size, id)) {
81         you_can = false;
82     }
83
84     return (you_can);
85 }
86
87 bool Map::canBulletWalkToThisPosition(Size &other_size, Occupant &shooter,
88                                         Occupant &target) {
89     bool you_can = true;
90
91     // if the object is stepping out of the map
92     if (map_size.areYouHalfOutSide(other_size))
93         you_can = false;
94
95     if (!areThisPointsEmpty(other_size, shooter, target)) {
96         you_can = false;
97     }
98
99     return (you_can);
100 }
101
102 bool Map::doesThisPositionExist(int x, int y) {
103     return map_size.areYouOnThisPoint(x,y);
104 }
105
106 bool Map::isThereLava(Size& other_size) {
107     int x_max, x_min, y_max, y_min;
108     other_size.calculateMaxAndMinForX(x_max, x_min);
109     other_size.calculateMaxAndMinForY(y_max, y_min);
110
111     int w_cell = terrain_map[0][0].getWidthOfCell();
112     // Check if any of the corners are stepping into lava
113     for (int y = y_min; y ≤ y_max; ++y) {
114         for (int x = x_min; x ≤ x_max; ++x) {
115             if (doesThisPositionExist(x,y)) {
116                 // Calculate the cell that holds this position
117                 int x_pos = x / w_cell;
118                 int y_pos = y / w_cell;
119
120                 if (terrain_map[x_pos][y_pos].getTerrainType() == "Lava") {
121                     return terrain_map[x_pos][y_pos].isThereACollision(
122                         other_size);
123                 }
124             }
125         }
126     }
127
128     return false;
129 }
130
131 bool Map::thereIsABridge(Size& other_size) {

```

jun 27, 17 14:46

map.cpp

Page 3/4

```

132     bool bridge = false;
133     for(auto x: *all_occupants) {
134         if(x->isThereACollision(other_size) ^ x->getType() == "Bridge") {
135             bridge = true;
136             break;
137         }
138     }
139     return bridge;
140 }
141
142 std::string &Map::get_map() {
143     return xml;
144 }
145
146 std::vector<Occupant *> &Map::getOccupants() {
147     return *this->all_occupants;
148 }
149
150 void Map::updateOccupants(std::vector<Occupant *> *all_occupants) {
151     *this->all_occupants = all_occupants;
152 }
153
154 Occupant* Map::checkForEnemiesOn(Size &range, Occupant& unit) {
155     for(auto x: *all_occupants) {
156         if(x->getId() != unit.getId() ^ x->isThereACollision(range)
157            ^ (types[x->getType()] == UNIT ^ types[x->getType()] == VEHICLE)
158            ^ x->getTeam() != "Neutral" ^ unit.getTeam() != x->getTeam()) {
159             return x;
160         }
161     }
162     return &unit;
163 }
164
165 void Map::buildTypeMap() {
166     types.insert(std::pair<std::string, std::string>("Fort", BUILDING));
167     types.insert(std::pair<std::string, std::string>
168         ("vehiculeFactory", BUILDING));
169     types.insert(std::pair<std::string, std::string>("robotFactory", BUILDING));
170     types.insert(std::pair<std::string, std::string>("Factory", BUILDING));
171     types.insert(std::pair<std::string, std::string>("Rock", NATURE));
172     types.insert(std::pair<std::string, std::string>("iceblock", NATURE));
173     types.insert(std::pair<std::string, std::string>("grunt", UNIT));
174     types.insert(std::pair<std::string, std::string>("Psycho", UNIT));
175     types.insert(std::pair<std::string, std::string>("Tough", UNIT));
176     types.insert(std::pair<std::string, std::string>("Pyro", UNIT));
177     types.insert(std::pair<std::string, std::string>("Sniper", UNIT));
178     types.insert(std::pair<std::string, std::string>("laser", UNIT));
179     types.insert(std::pair<std::string, std::string>("jeep", VEHICLE));
180     types.insert(std::pair<std::string, std::string>("MediumTank", VEHICLE));
181     types.insert(std::pair<std::string, std::string>("LightTank", VEHICLE));
182     types.insert(std::pair<std::string, std::string>("HeavyTank", VEHICLE));
183     types.insert(std::pair<std::string, std::string>("MML", VEHICLE));
184     types.insert(std::pair<std::string, std::string>("flag", FLAG));
185 }
186
187 bool Map::tellIfItIsGrabbable(std::string& type) {
188     std::string tmp = type;
189     return types[tmp] == VEHICLE;
190 }
191
192 bool Map::tellIfItIsBuilding(std::string &type) {
193     std::string tmp = type;
194     return types[tmp] == BUILDING;
195 }
196
197

```



jun 27, 17 14:46

map.cpp

Page 4/4

198  
199

jun 27, 17 14:46

main.cpp

Page 1/1

```

1  #include <iostream>
2  #include <pugixml.hpp>
3  #include "server.h"
4
5
6  unsigned int load_port(const char* cfg_file_path) {
7      pugi::xml_document doc;
8      pugi::xml_parse_result result = doc.load_file(cfg_file_path);
9      if (!result) {
10         std::cout << "Error reading cfg file: " << result.description()
11             << std::endl;
12         return 0;
13     }
14     pugi::xml_node port_node = doc.child("Config").child("Port");
15     std::string port = port_node.attribute("port").value();
16     if (!port.size()) {
17         std::cout << "Error reading port from cfg file!" << std::endl;
18         return 0;
19     }
20     int a = std::stoi(port);
21     return (unsigned int) a;
22 }
23
24 int main (int argc, char **argv) {
25     if (argc < 2) {
26         std::cout << "Usage: ./Z_Server <cfg file>" << std::endl;
27         return 1;
28     }
29
30     std::string cfg = argv[1];
31     if (!cfg.size()) {
32         std::cout << "Error loading cfg file" << std::endl;
33         return 1;
34     }
35     unsigned int port = load_port(argv[1]);
36     if (!port) {
37         return 1;
38     }
39
40     try {
41         Menu menu(cfg);
42
43         Server server_accepter(port, menu);
44         server_accepter.start();
45         char exit = 'a';
46
47         while (exit != 'q') {
48             std::cin >> exit;
49         }
50
51         server_accepter.stop();
52         server_accepter.join();
53
54         menu.shutdown();
55
56         return 0;
57     } catch (SocketError& e) {
58         std::cout << e.what();
59         return 1;
60     }
61 }
62

```

jun 27, 17 14:46

## lobby.h

Page 1/1

```

1  //
2  // Created by rodian on 29/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_LOBBY_H
6  #define Z_TPGRUPAL_LOBBY_H
7
8
9  #include "player.h"
10 #include "game.h"
11 #include "MapLoader.h"
12 class Game;
13 class Player;
14
15 class Lobby {
16 private:
17     int lobby_id;
18     bool all_ready, game_started;
19     std::vector<Player*> players;
20     Game* game;
21     std::vector<std::vector<std::string>> teams;
22     std::map<std::string, std::string> maps;
23     std::mutex &m;
24     std::string& config;
25 public:
26     Lobby(int id, std::string& config, std::mutex &m);
27
28     bool addPlayer(Player* player);
29
30     void startGame(const std::string& map_name);
31
32     void ready();
33
34     std::vector<std::string> get_player_names();
35
36     void unReady();
37
38     int get_id();
39
40     void exitLobby(Player* player);
41
42     std::string get_loaded_maps();
43
44     void load_maps();
45
46     void shutDown();
47
48     bool haveGameFinished();
49
50     void disconnectPlayer( Player *player);
51
52     ~Lobby();
53 };
54
55 #endif //Z_TPGRUPAL_LOBBY_H

```

jun 27, 17 14:46

## lobby.cpp

Page 1/3

```

1  //
2  // Created by rodian on 29/05/17.
3  //
4
5  #include <sstream>
6  #include "lobby.h"
7
8  Lobby::Lobby(int id, std::string& config, std::mutex &m) : lobby_id(id),
9                                                         config(config),
10                                                         all_ready(false),
11                                                         game_started(false),
12                                                         m(m){
13
14     load_maps();
15 }
16
17 void Lobby::startGame(const std::string& map_name) {
18     Lock l(m);
19     if(all_ready) {
20         //start game
21         game_started = true;
22         auto path_it = maps.find(map_name);
23         if (path_it != maps.end()) {
24             std::string path = path_it->second;
25
26             // build teams
27             std::vector<Team> teams_info;
28             for (int i = 0; i < teams.size(); ++i) {
29                 std::vector<PlayerInfo> playersInfo;
30                 for (int j = 0; j < teams[i].size(); ++j) {
31                     PlayerInfo new_player(teams[i][j]);
32                     for (auto p: players) {
33                         if (p->getId() == teams[i][j]) {
34                             new_player.addMessenger(p->getMessenger());
35                             playersInfo.push_back(new_player);
36                         }
37                     }
38                 }
39                 if (!playersInfo.empty()) {
40                     Team new_team(playersInfo, i);
41                     teams_info.push_back(new_team);
42                 }
43             }
44             for (auto p: players) {
45                 p->getInGame();
46             }
47
48             game = new Game(path, config, teams_info, players);
49             game->start();
50         }
51     }
52 }
53
54 void Lobby::ready() {
55     Lock l(m);
56     if (players.size() >= 2) {
57         bool any_not_ready = false;
58         for (auto p: players) {
59             if (!p->areYouReady()) {
60                 any_not_ready = true;
61             }
62         }
63         if (!any_not_ready)
64             all_ready = true;
65     }
66 }

```

jun 27, 17 14:46

lobby.cpp

Page 2/3

```

67
68 bool Lobby::addPlayer(Player* player) {
69     bool added = false;
70     if (!game_started) {
71         if (players.size() < 4) {
72             players.push_back(player);
73             added = true;
74         }
75
76         std::string names_cmd = "names-";
77         for (std::string name : get_player_names()) {
78             names_cmd += name + "-";
79         }
80
81         for (Player *p : players) {
82             p->getMessenger()->sendMessage(names_cmd);
83         }
84         teams.push_back(std::vector<std::string>());
85         teams.back().push_back(player->getId());
86         return added;
87     } else {
88         return added;
89     }
90 }
91
92 std::vector<std::string> Lobby::get_player_names() {
93     std::vector<std::string> names;
94     for (Player* p : players) {
95         names.push_back(p->getId());
96     }
97     return names;
98 }
99
100
101 int Lobby::get_id() {
102     return lobby_id;
103 }
104
105 void Lobby::unReady() {
106     Lock l(m);
107     all_ready = false;
108 }
109
110 void Lobby::exitLobby(Player *player) {
111     Lock l(m);
112     std::vector<Player *>::iterator it = players.begin();
113     for (; it != players.end(); ++it) {
114         if ((*it)->getId() == player->getId()) {
115             players.erase(it);
116             break;
117         }
118     }
119
120     for (auto& t: teams) {
121         std::vector<std::string>::iterator ito = t.begin();
122         for (; ito != t.end(); ++ito) {
123             if (*ito == player->getId()) {
124                 t.erase(ito);
125                 break;
126             }
127         }
128     }
129
130     std::string names_cmd = "names-";
131     for (std::string name : get_player_names()) {
132         names_cmd += name + "-";

```

jun 27, 17 14:46

lobby.cpp

Page 3/3

```

133     }
134
135     for (Player* p : players) {
136         p->getMessenger()->sendMessage(names_cmd);
137     }
138 }
139
140 std::string Lobby::get_loaded_maps() {
141     std::stringstream s;
142     s << "mapsinfo-";
143     for (auto map : maps) {
144         s << map.first << "-";
145     }
146     return s.str();
147 }
148
149
150 void Lobby::load_maps() {
151     pugi::xml_document doc;
152     doc.load_file(config.c_str());
153     pugi::xml_node cfg_root = doc.child("Config");
154     pugi::xml_node maps_node = cfg_root.child("Maps");
155     for (pugi::xml_node map : maps_node.children()) {
156         maps[map.attribute("name").value()] = map.attribute("path").value();
157     }
158 }
159
160 void Lobby::shutDown() {
161     game->shutDownGame();
162     game->join();
163 }
164
165 Lobby::~Lobby() {}
166
167 bool Lobby::haveGameFinished() {
168     if (game_started) {
169         return game->gameHaveFinished();
170     } else {
171         return false;
172     }
173 }
174
175 void Lobby::disconnectPlayer(Player *player) {
176     std::vector<Player*>::iterator it = players.begin();
177     for (; it != players.end(); ++it) {
178         if ((*it)->getId() == player->getId()) {
179             if (game_started)
180                 this->game->disconnectPlayer(player->getId());
181             players.erase(it);
182             break;
183         }
184     }
185 }

```

jun 27, 17 14:46

game.h

Page 1/2

```

1  //
2  // Created by rodian on 29/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_GAME_H
6  #define Z_TPGRUPAL_GAME_H
7
8
9  #include "map.h"
10 #include "controlUnit.h"
11 #include "player.h"
12 #include "commandMonitor.h"
13
14 class ControlUnit;
15 class Player;
16
17 class Game: public Thread {
18 private:
19     std::mutex m;
20     CommandMonitor commands;
21     std::map<int, Unit*> all_units;
22     std::vector<Player*> players;
23     std::vector<Occupant*> all_occupants;
24     std::vector<Territory*> territories;
25     std::vector<Team> teams;
26     ControlUnit* control;
27     std::shared_ptr<Map> map;
28     std::string path, config;
29     std::map<std::string, std::string> types;
30     std::vector<UnitMold*> unit_molds;
31     bool finished;
32
33 public:
34     // Game(std::vector<Player *> players, std::vector<Messenger *> msgr,
35     //      std::shared_ptr<Map> map, std::map<int, Unit *> units,
36     //      std::vector<Team>& teams_info, std::vector<Occupant *> occupants,
37     //      std::vector<Territory *> &territories);
38
39     Game(std::string path, std::string &config, std::vector<Team> &teams_info,
40         std::vector<Player *> &players);
41
42     void run();
43
44     void shutDownGame();
45
46     void sendMapInfo(ControlUnit &control);
47
48     void sendOccupantsInfo();
49
50     bool gameHaveFinished();
51
52     void disconnectPlayer(std::string player);
53
54     ~Game();
55 private:
56     void analyseOccupantsInfo(std::string& info);
57
58     void buildTypeMap();
59
60     void sincronizeOccupants();
61
62     void sendTerritoryInfo();
63
64     void createStartingUnits();
65
66     void buildMap();

```

jun 27, 17 14:46

game.h

Page 2/2

```

67
68     std::vector<Messenger*> getMessengers();
69 };
70
71
72 #endif //Z_TPGRUPAL_GAME_H

```

jun 27, 17 14:46

game.cpp

Page 1/4

```

1  //
2  // Created by rodian on 29/05/17.
3  //
4
5  #include <sstream>
6  #include <string>
7  #include "game.h"
8
9  Game::Game(std::string path, std::string &config, std::vector<Team> &teams_info,
10             std::vector<Player *> &players) :
11             commands(m), teams(teams_info), path(path), config(config),
12             players(players), finished(false) {}
13
14 void Game::run() {
15     this->buildMap();
16     this->sincronizeOccupants();
17     std::vector<Messenger*> messengers = getMessengers();
18     control = new ControlUnit(messengers, all_units, all_occupants,
19                             teams, commands, territories);
20     this->sendMapInfo(*control);
21     this->buildTypeMap();
22     this->sendTerritoryInfo();
23     this->sendOccupantsInfo();
24     control->run();
25
26     for (auto& m: unit_molds) {
27         delete(m);
28     }
29     finished = true;
30 }
31
32 void Game::buildMap() {
33     MapLoader maploader(path, config);
34     map = maploader.get_map();
35
36     all_occupants = map->getOccupants();
37
38     // add a Fortress to each player
39     std::vector<Factory*> forts = maploader.get_forts();
40     unit_molds = forts.back()->getMolds();
41     for (auto& t: teams) {
42         std::vector<PlayerInfo>& playersInfo = t.getPlayersInfo();
43         for (auto& p: playersInfo) {
44             Factory* fortress = forts.back();
45             fortress->changeTeam(p.getPlayerId());
46             // set changed boolean to false
47             fortress->haveYouChanged();
48             p.addFortress(fortress);
49             all_occupants.push_back((Occupant*) fortress);
50             forts.pop_back();
51         }
52     }
53     territories = maploader.get_territories();
54 }
55
56 void Game::shutDownGame() {
57     if (!finished)
58         control->finishGame();
59 }
60
61 void Game::sendOccupantsInfo() {
62     std::string info;
63     this->analyseOccupantsInfo(info);
64     for (auto& player : players) {
65         player->getMessenger()->sendMessage(info);
66     }

```

jun 27, 17 14:46

game.cpp

Page 2/4

```

67 }
68
69 void Game::sendMapInfo(ControlUnit &control) {
70     std::string& map_str = map.get()->get_map();
71     for (auto& player : players) {
72         std::string msg = "loadmap-" + map_str;
73         player->getMessenger()->sendMessage(msg);
74         player->addControlUnit(&control, &commands);
75     }
76 }
77
78 void Game::analyseOccupantsInfo(std::string& info) {
79     for (auto o: all_occupants) {
80         if (types[o->getType()] == "Nature") {
81             info += "addnature-";
82         } else if (types[o->getType()] == "Building") {
83             info += "addbuilding-";
84         } else if (types[o->getType()] == "Unit") {
85             info += "addunit-";
86         }
87         info += std::to_string(o->getId()) + "-";
88         Position pos = o->getPosition();
89         info += std::to_string(pos.getX()) + "-";
90         info += std::to_string(pos.getY()) + "-";
91         info += o->getType() + "-";
92         info += o->getTeam() + "-";
93         info += std::to_string(o->getLifeLeft()) + "|"; // life left == max hp
94     }
95 }
96
97 void Game::buildTypeMap() {
98     types.insert(std::pair<std::string, std::string>("Fort", "Building"));
99     types.insert(std::pair<std::string, std::string>
100                 ("vehiculeFactory", "Building"));
101     types.insert(std::pair<std::string, std::string>("robotFactory", "Building"));
102     types.insert(std::pair<std::string, std::string>("Factory", "Building"));
103     types.insert(std::pair<std::string, std::string>("Rock", "Nature"));
104     types.insert(std::pair<std::string, std::string>("iceblock", "Nature"));
105     types.insert(std::pair<std::string, std::string>("grunt", "Unit"));
106     types.insert(std::pair<std::string, std::string>("Psycho", "Unit"));
107     types.insert(std::pair<std::string, std::string>("Tough", "Unit"));
108     types.insert(std::pair<std::string, std::string>("Pyro", "Unit"));
109     types.insert(std::pair<std::string, std::string>("Sniper", "Unit"));
110     types.insert(std::pair<std::string, std::string>("laser", "Unit"));
111     types.insert(std::pair<std::string, std::string>("jeep", "Unit"));
112     types.insert(std::pair<std::string, std::string>("MediumTank", "Unit"));
113     types.insert(std::pair<std::string, std::string>("LightTank", "Unit"));
114     types.insert(std::pair<std::string, std::string>("HeavyTank", "Unit"));
115     types.insert(std::pair<std::string, std::string>("MML", "Unit"));
116 }
117
118 void Game::sincronizeOccupants() {
119     for (auto& t: territories) {
120         std::map<int, Factory*> factories = t->getFactories();
121         for (auto& f: factories) {
122             (f.second)->resetSelectedUnit();
123             all_occupants.push_back((Occupant*)(f.second));
124         }
125         for (auto& team: teams) {
126             std::vector<PlayerInfo>& players = team.getPlayersInfo();
127             for (auto& p: players) {
128                 Factory* fortress = p.getFortress();
129                 for (auto& f: factories) {
130                     if (fortress->getId() == f.second->getId()) {
131                         std::string new_team = p.getPlayerId();
132                         t->grabFlag(new_team);

```

jun 27, 17 14:46

game.cpp

Page 3/4

```

133         p.addTerritory(t);
134     }
135 }
136 }
137 }
138 }
139 map->updateOccupants(&all_occupants);
140 this->createStartingUnits();
141 }
142
143 void Game::sendTerritoryInfo() {
144     std::stringstream info;
145     for (auto& t: territories) {
146         info << "updateTerritory-";
147         info << std::to_string(t->getId()) << "-" << t->getTeam() << "-";
148         Position flag = t->getFlag()->getPosition();
149         info << std::to_string(flag.getX()) << "-" <<
150             std::to_string(flag.getY()) << "|";
151     }
152     for (auto& player : players) {
153         player->getMessenger()->sendMessage(info.str());
154     }
155 }
156
157 void Game::createStartingUnits() {
158     for (auto& team: teams) {
159         std::vector<PlayerInfo>& players = team.getPlayersInfo();
160         for (auto& p : players) {
161             Factory* fortress = p.getFortress();
162             int id_counter = (int)(territories.size() + all_occupants.size());
163             fortress->createStartingUnits(id_counter);
164             std::vector<Unit *> tmp = fortress->getUnits();
165             for (auto &u: tmp) {
166                 u->recalculateMyStartPosition();
167                 all_units[u->getId()] = u;
168                 all_occupants.push_back((Occupant*)u);
169                 // set changed boolean to false
170                 u->haveYouChanged();
171             }
172         }
173     }
174 }
175
176 std::vector<Messenger *> Game::getMessengers() {
177     std::vector<Messenger *> messengers;
178     for (auto& p: players) {
179         messengers.push_back(p->getMessenger());
180     }
181     return messengers;
182 }
183
184 Game::~Game() {
185     delete (this->control);
186 }
187
188 bool Game::gameHaveFinished() {
189     return finished;
190 }
191
192 void Game::disconnectPlayer(std::string id_player) {
193     for (auto& t: teams) {
194         std::vector<PlayerInfo>& players_info = t.getPlayersInfo();
195         std::vector<PlayerInfo>::iterator it = players_info.begin();
196         for (; it != players_info.end(); ++it) {
197             if ((*it).getPlayerId() == id_player) {
198                 players_info.erase(it);

```

jun 27, 17 14:46

game.cpp

Page 4/4

```

199         break;
200     }
201 }
202 }
203 std::vector<Player*>::iterator it = players.begin();
204 for(; it != players.end(); ++it) {
205     if ((*it)->getId() == id_player) {
206         players.erase(it);
207         break;
208     }
209 }
210 }

```

jun 27, 17 14:46

factory.h

Page 1/1

```

1 //
2 // Created by rodian on 22/05/17.
3 //
4
5 #ifndef Z_TPGRUPAL_FACTORY_H
6 #define Z_TPGRUPAL_FACTORY_H
7 #include "unitMold.h"
8 #include <memory>
9 class Factory: public Occupant {
10     bool running;
11     int tech_level,time_counter;
12     std::vector<UnitMold*> units;
13     std::vector<UnitMold*>::iterator mold;
14     std::vector<Unit*> new_units;
15     std::shared_ptr<Map> map;
16     std::map<std::string, Weapon> weapons;
17
18 public:
19     Factory(int id, int life, std::string& type, Size position,
20             std::vector<UnitMold*>& units, std::shared_ptr<Map> map,
21             std::map<std::string, Weapon> &weapons);
22
23     // starts the creation of the selected unit
24     void build(int& id_counter);
25
26     void startBuilding(const std::string& player_id);
27
28     // Returns the creational time of the selected Unit
29     int getSelectedUnitTime();
30
31     // Returns the type of the unit that now is selected
32     UnitMold* nextUnit();
33
34     UnitMold* previousUnit();
35
36     UnitMold * getSelectedUnit();
37
38     void changeTechLevel(int tech_level);
39
40     int getCreationSpeed();
41
42     bool haveNewUnits();
43
44     std::vector<Unit*> getUnits();
45
46     void resetSelectedUnit();
47
48     void createStartingUnits(int &id_counter);
49
50     std::vector<UnitMold*> getMolds();
51
52     ~Factory();
53 };
54
55 #endif //Z_TPGRUPAL_FACTORY_H
56
57

```

jun 27, 17 14:46

factory.cpp

Page 1/2

```

1 //
2 // Created by rodian on 22/05/17.
3 //
4
5 #include <memory>
6 #include "factory.h"
7 #define ID 3
8 Factory::Factory(int id, int life, std::string& type, Size position,
9                 std::vector<UnitMold*>& units, std::shared_ptr<Map> map,
10                 std::map<std::string, Weapon> &weapons) :
11     Occupant(id, life,type, position), running(false),time_counter(0), units(units),
12     map(map), weapons(weapons), tech_level(1){
13     mold = units.begin();
14 }
15
16 void Factory::build(int& id_counter) {
17     if (time_counter ≥ (*mold)→getCreationTime()) {
18         Size u_size = (*mold)→getUnitSize();
19         Position factory_pos = this→obj_size.getPosition();
20         u_size.moveTo(factory_pos.getX(),factory_pos.getY());
21         Weapon u_weapon = weapons.at((*mold)→getWeaponType());
22
23         for (int i = 0; i < (*mold)→getCreationQuantity(); ++i) {
24             Unit* new_unit = (*mold)→createUnit(
25                 id_counter,u_size,*map,u_weapon);
26             new_unit→changeTeam(this→team);
27             new_units.push_back(new_unit);
28             ++id_counter;
29         }
30         this→changed = true;
31         time_counter = 0;
32     } else if (running ^ time_counter < (*mold)→getCreationTime()) {
33         time_counter += 1 + tech_level;
34         this→changed = true;
35     } else {
36         this→changed = false;
37     }
38 }
39
40 int Factory::getSelectedUnitTime() {
41     return (*mold)→getCreationTime();
42 }
43
44 UnitMold* Factory::nextUnit() {
45     int i = 0;
46     this→running = false;
47     time_counter = 0;
48     while (i ≡ 0 ∨ (*mold)→getTechnologyLevel() > this→tech_level) {
49         ++mold;
50         if (mold ≡ units.end())
51             mold = units.begin();
52         ++i;
53     }
54     return *mold;
55 }
56
57 void Factory::changeTechLevel(int tech_level) {
58     this→tech_level = tech_level;
59 }
60
61 void Factory::startBuilding(const std::string &player_id) {
62     if (player_id ≡ this→getTeam())
63         running = true;
64 }
65
66 int Factory::getCreationSpeed() {

```

jun 27, 17 14:46

**factory.cpp**

Page 2/2

```

67     return (((*mold)→getCreationTime() - time_counter) / (1 + tech_level));
68 }
69
70 bool Factory::haveNewUnits() {
71     return (!this→new_units.empty());
72 }
73
74 std::vector<Unit*> Factory::getUnits() {
75     std::vector<Unit*> tmp = new_units;
76     new_units.clear();
77     return tmp;
78 }
79
80 UnitMold * Factory::getSelectedUnit() {
81     return *mold;
82 }
83
84 UnitMold* Factory::previousUnit() {
85     int i = 0;
86     this→running = false;
87     time_counter = 0;
88     while (i == 0 ∨ (*mold)→getTechnologyLevel() > this→tech_level) {
89         if (mold == units.begin()) {
90             mold = units.end();
91         }
92         --mold;
93         ++i;
94     }
95     return *mold;
96 }
97
98 void Factory::resetSelectedUnit() {
99     mold = units.begin();
100 }
101
102 void Factory::createStartingUnits(int &id_counter) {
103     time_counter = (*mold)→getCreationTime();
104     this→build(id_counter);
105 }
106
107 Factory::~Factory() {}
108
109 std::vector<UnitMold *> Factory::getMolds() {
110     return units;
111 }
112

```

jun 27, 17 14:46

**controlUnit.h**

Page 1/2

```

1  //
2  // Created by rodian on 22/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_CONTRONIT_H
6  #define Z_TPGRUPAL_CONTRONIT_H
7
8  #include <iostream>
9  #include <chrono>
10 #include <thread>
11 #include "unit.h"
12 #include "../common/Lock.h"
13 #include "../common/messenger.h"
14 #include "factory.h"
15 #include "territory.h"
16 #include "team.h"
17 #include "command.h"
18 #include "commandMonitor.h"
19
20 class Command;
21 class CommandMonitor;
22
23 class ControlUnit {
24 private:
25     std::map<int, Unit*> &all_units;
26     std::vector<Territory*> &territories;
27     std::vector<Occupant*> &all_occupants;
28     std::vector<Messenger*> players;
29     CommandMonitor &commands;
30     std::mutex m;
31     bool winning;
32     std::vector<Team> &teams;
33     std::vector<Bullet*> all_bullets;
34     std::vector<int> changed_units;
35     std::vector<Occupant> changed_occupants;
36     std::vector<Factory> changed_factories;
37     std::vector<Unit*> eliminated_units;
38     int objects_counter;
39
40 public:
41     ControlUnit(std::vector<Messenger *> &new_players,
42               std::map<int, Unit *> &all_units,
43               std::vector<Occupant *> &all_occupants,
44               std::vector<Team> &teams, CommandMonitor &commands,
45               std::vector<Territory *> &territories);
46
47     // Method to start checking commands from players
48     void run();
49
50     void sleepFor(std::chrono::duration<double> sec);
51
52     // Meant to make every unit make a micro action on the Tic
53     void unitsMakeMicroAction();
54
55     // Checks if any Occupant is dead. If so, it will remove it from the game
56     void checkAllLivingOccupants();
57
58     void moveAllBullets();
59
60     // Command move unit. Meant to give the order to the unit to start moving
61     // to de (x,y) position
62     void cmdMoveUnit(const std::string& id_player, int id, int x, int y);
63
64     void cmdAttack(const std::string& attacker_team, int id_unit, int target);
65
66     void cmdGrab(const std::string& id_player, int id_unit, int target);

```



jun 27, 17 14:46

controlUnit.h

Page 2/2

```

67     void cmdFactoryCreate(const std::string& player_id, int id_factory);
68
69     void cmdFactoryNext(const std::string& player_id, int id_factory);
70
71     void cmdFactoryPrev(const std::string& player_id, int id_factory);
72
73     void cmdFactoryCurrent(const std::string& player_id, int id_factory);
74
75     void finishGame();
76
77 private:
78     // Process all commands on commands vector and leaves the vector empty
79     void executeCommands();
80
81     void sendUpdateMessage();
82
83     void sendMessageTo(const std::string& player_id, std::string& msg);
84
85     std::string getUpdateInfo();
86
87     std::string getInfoFromUnit(Unit& unit);
88
89     std::string getInfoFromOccupant(Occupant& Occupant);
90
91     std::string getInfoFromBullets(Bullet& bullet);
92
93     std::string getInfoFromFactories(Factory& factory);
94
95     std::string getInfoFromUnitMold(UnitMold& mold, double time);
96
97     std::string getInfoForAddUnit(Unit& unit);
98
99     void makeTerritoriesChecks();
100
101     void makeFactoryChecks();
102
103     void checkForWinner();
104
105     void sendFinnalMessage();
106
107     void getTime(int& minutes, int& seconds, double time);
108
109     void freeMemory();
110 };
111
112
113 #endif //Z_TPGRUPAL_CONTRONUNIT_H
114

```

jun 27, 17 14:46

controlUnit.cpp

Page 1/9

```

1  //
2  // Created by rodian on 22/05/17.
3  //
4
5  #include "controlUnit.h"
6  #define WAIT 0.3
7  #define FLAG "flag"
8  #define NATURE "Rock"
9
10 ControlUnit::ControlUnit(std::vector<Messenger *> &new_players,
11                          std::map<int, Unit *> &all_units,
12                          std::vector<Occupant *> &occupants,
13                          std::vector<Team> &teams, CommandMonitor &commands,
14                          std::vector<Territory *> &territories) :
15     all_units(all_units), territories(territories),
16     all_occupants(occupants), players(new_players), commands(commands),
17     winning(false), teams(teams) {
18 }
19
20 void ControlUnit::run() {
21     objects_counter = (int)all_occupants.size();
22     while(!winning) {
23         std::chrono::duration<double> t3(WAIT);
24
25         auto t1 = std::chrono::high_resolution_clock::now();
26
27         // execute commands
28         executeCommands();
29
30         // do stuff
31         this->moveAllBullets();
32         this->unitsMakeMicroAction();
33         this->makeTerritoriesChecks();
34         this->checkAllLivingOccupants();
35
36         //send update message
37         this->sendUpdateMessage();
38
39         this->checkForWinner();
40
41         auto t2 = std::chrono::high_resolution_clock::now();
42         std::chrono::duration<double> time_span = t3 - (t2 - t1);
43         sleepFor(time_span);
44         changed_units.clear();
45         changed_occupants.clear();
46         changed_factories.clear();
47         for (auto& u: eliminated_units) {
48             delete(u);
49         }
50         eliminated_units.clear();
51     }
52     // send victory or defeated message
53     this->sendFinnalMessage();
54     this->freeMemory();
55 }
56
57 void ControlUnit::sleepFor(std::chrono::duration<double> msec) {
58     std::this_thread::sleep_for((msec));
59 }
60
61 void ControlUnit::unitsMakeMicroAction() {
62     // erase units with life 0
63     std::vector<int> units_id;
64     for (auto& x: all_units) {
65         Unit& unit = *x.second;
66         if (unit.doesNeedToDisappear()) {

```

jun 27, 17 14:46

controlUnit.cpp

Page 2/9

```

67         units_id.push_back(x.first);
68     }
69 }
70 for (auto& id: units_id) {
71     eliminated_units.push_back(all_units[id]);
72     all_units.erase(id);
73 }
74
75 // units alive make micro action
76 for (auto& x: all_units) {
77     Unit& unit = *x.second;
78     // check if someone changed the unit
79     bool was_changed = false;
80     if (unit.haveYouChanged()) {
81         changed_units.push_back(unit.getId());
82         was_changed = true;
83     }
84     unit.makeAction();
85     // check if the unit changed
86     if (unit.haveYouChanged() ^ !was_changed) {
87         changed_units.push_back(unit.getId());
88     }
89     if (!unit.areYouAlive()) {
90         unit.mustDisappear();
91     } else if (unit.doYouHaveAnyBullets()) {
92         std::vector<Bullet*> tmp = unit.collectBullets();
93         for (auto& b: tmp) {
94             b->setCorrectId(objects_counter);
95             ++objects_counter;
96         }
97         all_bullets.insert(all_bullets.end(), tmp.begin(), tmp.end());
98     }
99 }
100 }
101
102 void ControlUnit::checkAllLivingOccupants() {
103     std::vector<Occupant*>::iterator it = all_occupants.begin();
104     for (; it != all_occupants.end(); ++it) {
105         if ((*it)->doYouNeedToDisappear()) {
106             //erase it from map
107             if ((*it)->getType() == NATURE) {
108                 delete((*it));
109             }
110             it = all_occupants.erase(it);
111             // if building put ruins
112         } else {
113             if ((*it)->haveYouChanged())
114                 changed_occupants.push_back((*it));
115             if (!(*it)->areYouAlive()) {
116                 (*it)->mustDisappear();
117             }
118             ++it;
119         }
120     }
121 }
122
123 void ControlUnit::makeTerritoriesChecks() {
124     for (auto& t: territories) {
125         if (t->doesTerritorysOwnerChanged()) {
126             std::string info = "update territory-";
127             info += std::to_string(t->getId()) + "-" + t->getTeam() + "-";
128             Position flag = t->getFlag()->getPosition();
129             info += std::to_string(flag.getX()) + "-" +
130                 std::to_string(flag.getY());
131             for (auto& team: teams) {

```

jun 27, 17 14:46

controlUnit.cpp

Page 3/9

```

133         std::vector<PlayerInfo*> players = team.getPlayersInfo();
134         for (auto& p: players) {
135             // the last owner must eliminate the territory from his
136             // vector
137             p->eliminateThisTerritory(t);
138
139             // the new owner must add it
140             if (t->getTeam() == p->getPlayerId()) {
141                 p->addTerritory(t);
142             }
143         }
144     }
145     for (auto y: players) {
146         y->sendMessage(info);
147     }
148 }
149
150 makeFactoryChecks();
151 }
152
153 void ControlUnit::makeFactoryChecks() {
154     for (auto t: territories) {
155         std::map<int, Factory*>& factories = t->getFactories();
156         auto it = factories.begin();
157         // vector to know witch factories erase
158         std::vector<int> factories_id;
159         for (; it != factories.end(); ++it) {
160             Factory *f = it->second;
161             bool was_changed = false;
162             if (f->haveYouChanged()) {
163                 changed_factories.push_back(*f);
164                 was_changed = true;
165             }
166             if (f->areYouAlive()) {
167                 f->build(objects_counter);
168                 // check if the factory changed
169                 if (f->haveYouChanged() ^ !was_changed) {
170                     changed_factories.push_back(*f);
171                 }
172                 if (f->haveNewUnits()) {
173                     std::vector<Unit*> tmp = f->getUnits();
174                     std::string msg = "";
175                     for (auto &u: tmp) {
176                         u->recalculateMyStartPosition();
177                         all_units[u->getId()] = u;
178                         all_occupants.push_back((Occupant*)u);
179                         // set changed boolean to false
180                         u->haveYouChanged();
181                         msg += "addunit-";
182                         msg += getInfoForAddUnit(*u);
183                     }
184                     for (auto y: players) {
185                         y->sendMessage(msg);
186                     }
187                 }
188             }
189             ++it;
190         }
191         for (auto& fact: factories_id) {
192             factories.erase(fact);
193         }
194     }
195 }
196
197 void ControlUnit::cmdMoveUnit(const std::string& id_player, int id, int x,
198                             int y) {

```

jun 27, 17 14:46

controlUnit.cpp

Page 4/9

```

199     std::map<int,Unit*>::iterator it;
200     it = all_units.find(id);
201     if ((*it->second).getTeam() == id_player)
202         (*it->second).calculateRoadTo(x,y);
203 }
204
205 void ControlUnit::cmdAttack(const std::string& attacker_team, int id_unit,
206                             int target) {
207     std::map<int,Unit*>::iterator it;
208     it = all_units.find(id_unit);
209     if ((*it->second).getTeam() == attacker_team) {
210         for (auto z: all_occupants) {
211             if (z->getId() == target) {
212                 if (z->getTeam() != attacker_team) {
213                     (*it->second).setTargetToAttack(z);
214                     break;
215                 }
216             }
217         }
218     }
219 }
220
221 void ControlUnit::cmdGrab(const std::string &id_player, int id_unit,
222                           int target) {
223     std::map<int,Unit*>::iterator it;
224     it = all_units.find(id_unit);
225     Unit& unit = (*it->second);
226     bool found = false;
227     if (unit.getTeam() == id_player) {
228         for (auto t: territories) {
229             if (t->getId() == target) {
230                 unit.setTargetToGrab(t->getFlag(),FLAG);
231                 found = true;
232             }
233         }
234     }
235     if (!found) {
236         for (auto& z: all_occupants) {
237             if (z->getId() == target) {
238                 unit.setTargetToGrab(z, z->getType());
239             }
240         }
241     }
242 }
243
244 void ControlUnit::cmdFactoryCreate(const std::string& player_id,
245                                    int id_factory) {
246     for (auto t: territories) {
247         std::map<int, Factory *> &factories = t->getFactories();
248         for (auto& f: factories) {
249             if (f.first == id_factory ^ f.second->areYouAlive()) {
250                 f.second->startBuilding(player_id);
251             }
252         }
253     }
254 }
255
256 void ControlUnit::cmdFactoryNext(const std::string &player_id, int id_factory) {
257     std::string info = "";
258     for (auto t: territories) {
259         std::map<int, Factory *> &factories = t->getFactories();
260         for (auto& f: factories) {
261             if (f.first == id_factory ^ f.second->getTeam() == player_id
262                 ^ f.second->areYouAlive()) {
263                 UnitMold* mold = f.second->nextUnit();
264                 info += "factorystats-";

```

jun 27, 17 14:46

controlUnit.cpp

Page 5/9

```

265         int creation_time = f.second->getCreationSpeed();
266         info += getInfoFromUnitMold(*mold,creation_time);
267         break;
268     }
269 }
270
271     sendMessageTo(player_id,info);
272 }
273
274 void ControlUnit::cmdFactoryPrev(const std::string &player_id, int id_factory) {
275     std::string info = "";
276     for (auto t: territories) {
277         std::map<int, Factory *> &factories = t->getFactories();
278         for (auto& f: factories) {
279             if (f.first == id_factory ^ f.second->getTeam() == player_id
280                 ^ f.second->areYouAlive()) {
281                 UnitMold* mold = f.second->previousUnit();
282                 info += "factorystats-";
283                 int creation_time = f.second->getCreationSpeed();
284                 info += getInfoFromUnitMold(*mold,creation_time);
285                 break;
286             }
287         }
288     }
289     sendMessageTo(player_id,info);
290 }
291
292 void ControlUnit::cmdFactoryCurrent(const std::string &player_id,
293                                    int id_factory) {
294     std::string info = "";
295     for (auto t: territories) {
296         std::map<int, Factory *> &factories = t->getFactories();
297         for (auto& f: factories) {
298             if (f.first == id_factory ^ f.second->areYouAlive()) {
299                 UnitMold* mold = f.second->getSelectedUnit();
300                 info += "factorystats-";
301                 int creation_time = f.second->getCreationSpeed();
302                 info += getInfoFromUnitMold(*mold,creation_time);
303                 break;
304             }
305         }
306     }
307     sendMessageTo(player_id,info);
308 }
309
310 void ControlUnit::executeCommands() {
311     std::vector<Command> commands_copy;
312     commands.copyCommands(commands_copy);
313
314     // Execute command
315     for (auto cmd: commands_copy) {
316         cmd();
317     }
318 }
319
320 void ControlUnit::sendMessageTo(const std::string& player_id,
321                                 std::string& msg) {
322     bool found = false;
323     for (auto& t: teams) {
324         std::vector<PlayerInfo>& plyrs = t.getPlayersInfo();
325         for (auto& p: plyrs) {
326             if (p.getPlayerId() == player_id) {
327                 p.getMessenger()->sendMessage(msg);
328                 found = true;

```

jun 27, 17 14:46

controlUnit.cpp

Page 6/9

```

331         break;
332     }
333 }
334     if (found)
335         break;
336 }
337 }
338
339 void ControlUnit::sendUpdateMessage() {
340     std::string info = getUpdateInfo();
341     if (!info.size()) {
342         return;
343     }
344     for (auto y: players) {
345         y->sendMessage(info);
346     }
347 }
348
349 std::string ControlUnit::getUpdateInfo() {
350     std::string update_msg = "";
351     for (auto z: changed_units) {
352         update_msg += "updateunit-";
353         update_msg += getInfoFromUnit(*all_units.at(z));
354     }
355
356     for (auto y: changed_occupants) {
357         update_msg += "updateoccupant-";
358         update_msg += getInfoFromOccupant(y);
359     }
360
361     for (auto& f: changed_factories) {
362         update_msg += "updatefactory-";
363         update_msg += getInfoFromFactories(f);
364     }
365
366     // for (auto b: all_bullets) {
367     //     update_msg += "updatebullet-";
368     //     update_msg += getInfoFromBullets(*b);
369     // }
370
371     return update_msg;
372 }
373
374 std::string ControlUnit::getInfoFromUnit(Unit &unit) {
375     std::string info = "";
376     info += std::to_string(unit.getId()) + "-";
377     info += unit.getActionState() + "-";
378     info += std::to_string(unit.getCurrentPosition().getX()) + "-";
379     info += std::to_string(unit.getCurrentPosition().getY()) + "-";
380     info += std::to_string(unit.getLifeLeft()) + "-";
381     info += unit.getTeam() + "|";
382     return info;
383 }
384
385 std::string ControlUnit::getInfoFromOccupant(Occupant& Occupant) {
386     std::string info = "";
387     info += std::to_string(Occupant.getId()) + "-";
388     info += std::to_string(Occupant.getPosition().getX()) + "-";
389     info += std::to_string(Occupant.getPosition().getY()) + "-";
390     info += std::to_string(Occupant.getLifeLeft()) + "|";
391     return info;
392 }
393
394 std::string ControlUnit::getInfoFromFactories(Factory &factory) {
395     std::string info = "";
396     info += std::to_string(factory.getId()) + "-";

```

jun 27, 17 14:46

controlUnit.cpp

Page 7/9

```

397     // This is the time needed before the next unit is build in seconds
398     double time = WAIT * factory.getCreationSpeed();
399     int min = 0, sec = 0;
400     getTime(min, sec, time);
401     info += std::to_string(min) + "-" + std::to_string(sec) + "-";
402     info += std::to_string(factory.getLifeLeft()) + "-";
403     info += factory.getTeam() + "|";
404     return info;
405 }
406
407 std::string ControlUnit::getInfoFromBullets(Bullet &bullet) {
408     std::string info = "";
409     info += bullet.getType() + "-";
410     info += std::to_string(bullet.getId()) + "-";
411     info += std::to_string(bullet.getPosition().getX()) + "-";
412     info += std::to_string(bullet.getPosition().getY()) + "|";
413     return info;
414 }
415
416 std::string ControlUnit::getInfoFromUnitMold(UnitMold &mold,
417     double creation_time) {
418     std::string info = "";
419     info += mold.getTypeOfUnit() + "-";
420     info += std::to_string(mold.getFireRate()) + "-";
421     double time = WAIT * creation_time;
422     int min = 0, sec = 0;
423     getTime(min, sec, time);
424     info += std::to_string(min) + "-" + std::to_string(sec) + "-";
425     info += std::to_string(mold.getLife()) + "|";
426     return info;
427 }
428
429 std::string ControlUnit::getInfoForAddUnit(Unit &unit) {
430     std::string info = "";
431     info += std::to_string(unit.getId()) + "-";
432     Position pos = unit.getPosition();
433     info += std::to_string(pos.getX()) + "-";
434     info += std::to_string(pos.getY()) + "-";
435     info += unit.getType() + "-";
436     info += unit.getTeam() + "-";
437     info += std::to_string(unit.getLifeLeft()) + "|";
438     return info;
439 }
440
441 void ControlUnit::moveAllBullets() {
442     std::vector<Bullet*>::iterator it = all_bullets.begin();
443     for (; it != all_bullets.end(); ) {
444         (*it)->move();
445         if ((*it)->doYouHaveToDisapear()) {
446             delete((*it));
447             it = all_bullets.erase(it);
448         } else {
449             if ((*it)->didHit())
450                 (*it)->disapear();
451             ++it;
452         }
453     }
454 }
455
456 void ControlUnit::checkForWinner() {
457     int teams_alive = 0;
458     for (auto t: teams) {
459         if (!t.doesTeamLose()) {
460             teams_alive += 1;
461         } else {

```

jun 27, 17 14:46

controlUnit.cpp

Page 8/9

```

463         std::vector<PlayerInfo>& losers = t.getPlayersInfo();
464         for (auto& w: losers) {
465             w.getMessenger()→sendMessage("loseryousuck");
466         }
467     }
468 }
469
470 if (teams_alive == 1) {
471     winning = true;
472 }
473 }
474
475 void ControlUnit::sendFinnalMessage() {
476     std::string winner = "winner-";
477     for (auto& t: teams) {
478         if (!t.doesTeamLose()) {
479             std::vector<PlayerInfo>& winners = t.getPlayersInfo();
480             for (auto& w: winners) {
481                 w.getMessenger()→sendMessage("winner");
482             }
483         } else {
484             std::vector<PlayerInfo>& losers = t.getPlayersInfo();
485             for (auto& w: losers) {
486                 w.getMessenger()→sendMessage("loseryousuck");
487             }
488         }
489     }
490 }
491
492 void ControlUnit::getTime(int &minutes, int &seconds, double time) {
493     double min = time/60;
494     minutes = (int)min;
495     double sec = min - minutes;
496     sec = sec * 60;
497     seconds = (int) sec;
498 }
499
500 void ControlUnit::freeMemory() {
501     // free memory
502     std::vector<Occupant*>::iterator it = all_occupants.begin();
503     for (; it != all_occupants.end(); ){
504         if ((*it)→getType() == NATURE) {
505             delete((*it));
506         }
507         it = all_occupants.erase(it);
508     }
509
510     for (auto& u: all_units) {
511         delete(u.second);
512     }
513     all_units.clear();
514
515     for (auto& t: territories) {
516         delete(t);
517     }
518
519     for (auto& b: all_bullets) {
520         delete(b);
521     }
522     all_bullets.clear();
523 }
524
525 void ControlUnit::finishGame() {
526     winning = true;
527 }
528

```

jun 27, 17 14:46

controlUnit.cpp

Page 9/9

529

jun 27, 17 14:46

compass.h

Page 1/3

```

1  //
2  // Created by rodian on 14/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_COMPASS_H
6  #define Z_TPGRUPAL_COMPASS_H
7
8  #include <iostream>
9  #include <vector>
10 #include <map>
11 #include "cell.h"
12 #include "node.h"
13 #include "map.h"
14
15 class Map;
16
17 // class Compass so every moving unit knows the fastest way to destiny
18 class Compass {
19 private:
20     Map& map;
21     std::vector<std::vector<Node*>> astar_map;
22     std::vector<Node*> closed_nodes;
23     std::vector<Node*> open_nodes;
24     std::vector<Position> road;
25     std::map<std::string,int> terrain_modifier;
26     int unit_id, unit_speed;
27     Size unit_size;
28     bool finished, clear;
29     Position destiny;
30
31 public:
32     // The Compass receives the map of Cells for calculations and the
33     // basic unit speed
34     Compass(Map &map, Size &unit_size, int unit_id, int unit_speed);
35
36     // Receives the current position of the unit and the destiny
37     // Returns a vector of Cells with the fastest way
38     std::vector<Position> getFastestWay(Position& from, Position& to);
39
40     // Returns true if the position is empty
41     bool canIWalkToThisPosition(Size& size);
42
43     // Returns true if the position is empty except for the occupant
44     bool canBulletWalkToThisPosition(Size& size, Occupant &shooter,
45                                     Occupant& occupant);
46
47     // Returns the Terrain factor on the (x,y) position
48     double getTerrainFactorOn(int x, int y);
49
50     void changeUnitSize(Size& unit_size);
51
52     void changeUnitSpeed(int speed);
53
54     void changeUnitId(int id);
55
56     void clearCompass();
57
58     bool checkIfItIsGrabbable(std::string& type) const;
59
60     bool checkIfItIsABuilding(std::string& type) const;
61
62     Occupant* checkForEnemiesOnRange(Occupant& unit, Size &range);
63
64     // Builds a Node map with the size of the original map
65     void buildNodeMap();
66

```

jun 27, 17 14:46

compass.h

Page 2/3

```

67     // Returns the position of destiny. If destiny is not a valid position
68     // it returns the closest valid position
69     Position getAValidPositionForDestiny(Position& destiny);
70
71     ~Compass();
72
73 private:
74
75     void setTerrainModifier();
76     // Writes the H value on every node of astar_map for the received position
77     // It use Manhattan distance
78     void setHValueForDestiny(Position& to);
79
80     // Only valid for Manhattan distance.
81     // Returns true if other is a diagonal node of reference
82     // otherwise false. Reference and other must be adjacent.
83     bool isThisNodeOnDiagonal(Node* reference, Node* other);
84
85     // Put the adyacents nodes that can be walk to on the open_nodes vector
86     void getAdjacents(Node* node, int step);
87
88     // Returns true if node and other are different nodes. Else, false
89     bool isNotMe(Node* node, Node* other);
90
91     // Puts the node in the correct order.
92     // The node with lower F will be on the back
93     void addToOpenInOrder(Node* node);
94
95     // If adj node hasn't been seen or the g value from ref node is lower
96     // than previous, it chages g value and the parent.
97     bool writeGandSetParent(Node *ref, Node *adj, int walk, int steps);
98
99     // Changes the position of the node
100    void changeNodePosition(Node* node);
101
102    // Inserts the node on the correct position
103    void insertNodeOnOpen(Node* node);
104
105    void getRoad(Position& from, Node* destiny);
106
107    // Returns the closest node on the closed_nodes vector
108    Node* searchForClosestNode();
109
110    // Returns a positive value of the result of x - y.
111    int getModuleOfSubtraction(int x, int y);
112
113    // checks the Neighbor nodes to see if destiny is among them
114    void checkIfIsDestinyNeighbor(Node *new_node, int step);
115
116    // Returns the closest valid position to pos
117    Position getClosestValidPosition(Position& pos);
118
119    void addPositions(Position& position);
120
121    void manageSteps(int& step, Position& start,
122                    Position& current_pos, Position& to);
123
124    void setHValueOnNode(Node* node);
125
126    void addPositionsInOrder(bool increase_x, bool increase_y, int x_max,
127                            int x_min, int y_max, int y_min);
128
129    int getModule(int x, int y);
130 };
131
132

```

jun 27, 17 14:46

compass.h

Page 3/3

```
133 #endif //Z_TPGRUPAL_COMPASS_H
```

jun 27, 17 14:46

compass.cpp

Page 1/11

```
1 //
2 // Created by rodian on 14/05/17.
3 //
4
5 #include "compass.h"
6
7 #define SIDEWALK 10
8 #define DIAGONALWALK 14
9 #define HMIN 100
10 #define STEP 2
11 #define CLOSERAREA 32
12 #define MIDDLEAREA 120
13 #define MAXLOOP 800
14
15 Compass::Compass(Map &map, Size &unit_size, int unit_id, int unit_speed)
16 : map(map),
17   unit_size(unit_size), unit_id(unit_id), unit_speed(unit_speed),
18   destiny(0,0), clear(true){
19     this->buildNodeMap();
20     this->setTerrainModifier();
21 }
22
23 void Compass::setTerrainModifier() {
24     terrain_modifier.insert(std::pair<std::string,int>("Carretera",1));
25     terrain_modifier.insert(std::pair<std::string,int>("Camino Asfaltado",1));
26     terrain_modifier.insert(std::pair<std::string,int>("Tierra",2));
27     terrain_modifier.insert(std::pair<std::string,int>("Pradera",2));
28     terrain_modifier.insert(std::pair<std::string,int>("Nieve",2));
29     terrain_modifier.insert(std::pair<std::string,int>("Agua", 10));
30 }
31
32 void Compass::buildNodeMap() {
33     // the nodes has the size of the unit that is using this compass
34     for(int it = 0; it < map.getWidth(); ++it) {
35         std::vector<Node*> row_vec;
36         astar_map.push_back(row_vec);
37         for(int jt = 0; jt < map.getHeigth(); ++jt) {
38             astar_map.back().push_back(new Node(it, jt,
39                                                 unit_size.getWidth(), unit_size.getHeight()));
40         }
41     }
42 }
43
44 std::vector<Position> Compass::getFastestWay(Position& from, Position& to) {
45     if (!clear)
46         clearCompass();
47     // check if it's a possible position
48     destiny = getAValidPositionForDestiny(to);
49     // if I'm already on the closest position return it
50     if (from.getX() == destiny.getX() ^ from.getY() == destiny.getY()) {
51         this->road.push_back(destiny);
52         return road;
53     } else {
54         // start algorithm
55         // add "from" to visited list
56         Node *start_node = astar_map[from.getX()][from.getY()];
57         std::string terrain_type = map.getTerrainType(from.getX(), from.getY());
58         start_node->setGValue(0, terrain_modifier[terrain_type]);
59         start_node->setNewParent(start_node);
60         Position start_pos = start_node->getPosition();
61         this->closed_nodes.push_back(start_node);
62         clear = false;
63
64         Node *closer_node = start_node;
65         // While haven't reach destiny node or open_nodes has nodes to visit.
66         finished = false;
```

jun 27, 17 14:46

compass.cpp

Page 2/11

```

67     bool open_nodes_empty = false;
68
69     int step = 1;
70     int step_check = step;
71     int i = 0;
72     while (!finished ^ (!open_nodes_empty)) {
73         // get adjacent's and add them to looking list in order of F value.
74         // On tie use H value.
75
76         this->getAdjacents(closer_node, step);
77
78         // if there are no adjacent's and open_node is empty, end search
79         if (open_nodes.empty()) {
80             open_nodes_empty = true;
81         } else {
82             // get the minimum F and add it to visit list
83             // (remove from looking list)
84             closer_node = open_nodes.back();
85             open_nodes.pop_back();
86             this->closed_nodes.push_back(closer_node);
87             Position cls_pos = closer_node->getPosition();
88             // check if destiny is between them
89
90             if (closed_nodes.back()->getHvalue() == 0)
91                 finished = true;
92
93             if (!finished)
94                 manageSteps(step, start_pos, cls_pos,
95                             destiny);
96         }
97
98         if (step_check != step) {
99             step_check = step;
100         }
101         ++i;
102         if (i >= MAXLOOP) {
103             finished = true;
104             closer_node = start_node;
105         }
106     }
107     Node *closest;
108     if (finished) {
109         this->getRoad(from, closer_node);
110     } else {
111         closest = this->searchForClosestNode();
112         this->getRoad(from, closest);
113     }
114     finished = false;
115     return road;
116 }
117
118 void Compass::setHValueForDestiny(Position& to) {
119     astar_map[to.getX()][to.getY()]>setHValue(0);
120
121     for (auto x: astar_map) {
122         for (auto y: x) {
123             Position tmp = y->getPosition();
124             int h_value = HMIN * (this->getModuleOfSubtraction(tmp.getX(),
125                 to.getX()) + this->getModuleOfSubtraction(tmp.getY(), to.getY()));
126             y->setHValue(h_value);
127         }
128     }
129 }
130
131 void Compass::getAdjacents(Node *node, int step) {

```

jun 27, 17 14:46

compass.cpp

Page 3/11

```

133     // get limits
134     int x_min = node->getPosition().getX() - step;
135     int x_max = node->getPosition().getX() + step;
136     int y_min = node->getPosition().getY() - step;
137     int y_max = node->getPosition().getY() + step;
138
139     bool adj_new_g;
140     Node* adj;
141     for (int x_pos = x_min; x_pos <= x_max; x_pos += step) {
142         for (int y_pos = y_min; y_pos <= y_max; y_pos += step) {
143             if (map.doesThisPositionExist(x_pos, y_pos)) {
144                 adj = astar_map[x_pos][y_pos];
145                 Size size = adj->getSize();
146
147                 // Check if whether node fit or the position is not available.
148                 // Also discard the node looking for his adjacent
149                 if ((map.canIWalkToThisPosition(size, unit_id)) ^
150                     this->isNotMe(node, adj)) {
151                     this->setHValueOnNode(adj);
152                     // G value differs when the node is diagonal or next to it
153                     if (this->isThisNodeOnDiagonal(node, adj)) {
154                         adj_new_g = this->writeGandSetParent(node, adj,
155                             DIAGONALWALK, 0);
156                     } else {
157                         adj_new_g = this->writeGandSetParent(node, adj,
158                             SIDEWALK, 0);
159                     }
160                     if (adj_new_g)
161                         this->addToOpenInOrder(adj);
162                 }
163             }
164             if (finished)
165                 break;
166         }
167         if (finished)
168             break;
169     }
170 }
171
172 bool Compass::isThisNodeOnDiagonal(Node* ref, Node* other) {
173     Position pos_ref = ref->getPosition();
174     Position pos_other = other->getPosition();
175     int diff_y = getModuleOfSubtraction(pos_ref.getY(), pos_other.getY());
176     int diff_x = getModuleOfSubtraction(pos_ref.getX(), pos_other.getX());
177     return ((diff_x > 0) ^ (diff_y > 0));
178 }
179
180 bool Compass::isNotMe(Node* node, Node* other) {
181     Position ref = node->getPosition();
182     Position ady = other->getPosition();
183     return !((ref.getX() == ady.getX()) ^ (ref.getY() == ady.getY()));
184 }
185
186 void Compass::addToOpenInOrder(Node* new_node) {
187     // Only add to the vector those that haven't been seen
188     if (!new_node->beenSeen()) {
189         this->insertNodeOnOpen(new_node);
190     } else {
191         this->changeNodePosition(new_node);
192     }
193 }
194
195 bool Compass::writeGandSetParent(Node *ref, Node *adj, int walk, int steps) {
196     Size adj_size = adj->getSize();
197     //calculate new g

```



jun 27, 17 14:46

compass.cpp

Page 4/11

```

199     int new_g = walk + ref->getGValue();
200     // get additional g for all steps
201     road.push_back(adj->getPosition());
202     Position actual = ref->getPosition();
203     addPositions(actual);
204     for (auto& pos: road) {
205         std::string terrain_type = map.getTerrainType(pos.getX(), pos.getY());
206         // when is a vehicle and it's water, don't add it to open list
207         if (!(unit_speed != 2 ^ terrain_type == "Agua" ^
208             !map.thereIsABridge(adj_size))) {
209             new_g += terrain_modifier[terrain_type];
210         } else {
211             new_g += (terrain_modifier[terrain_type] * 20);
212         }
213     }
214     road.clear();
215
216     bool adj_change_g = false;
217     // if F value from node is lower than previous or this
218     // adjacent hasn't been seen yet,
219     // add new g value and change parent.
220     Position pos = adj->getPosition();
221     std::string terrain_type = map.getTerrainType(pos.getX(), pos.getY());
222     // when is a vehicle and it's water, don't add it to open list
223     if (!(unit_speed != 2 ^ terrain_type == "Agua" ^
224         !map.thereIsABridge(adj_size))) {
225         int terrain_factor = terrain_modifier[terrain_type];
226         if ((adj->beenSeen() ^
227             (adj->getFValueIfGWere(new_g, terrain_factor) <
228                 adj->getFValue())) ^
229             (!adj->beenSeen())) {
230             adj->setGValue(new_g, terrain_factor);
231             adj->setNewParent(ref);
232             adj_change_g = true;
233         }
234     }
235     return adj_change_g;
236 }
237
238 void Compass::changeNodePosition(Node *node) {
239     // first erase node from vector
240     bool erased = false;
241     Position node_pos = node->getPosition();
242     std::vector<Node *>::iterator it = open_nodes.begin();
243     while ((!erased) ^ (it != open_nodes.end())) {
244         Position it_pos = (*it)->getPosition();
245         if ((it_pos.getX() == node_pos.getX() ^
246             (it_pos.getY() == node_pos.getY())) {
247             it = open_nodes.erase(it);
248             erased = true;
249         } else {
250             ++it;
251         }
252     }
253     // Add it again in correct position
254     this->insertNodeOnOpen(node);
255 }
256
257 void Compass::insertNodeOnOpen(Node *new_node) {
258     if (new_node->getHvalue() == 0) {
259         open_nodes.push_back(new_node);
260         finished = true;
261     } else {
262         bool inserted = false;
263         // Save nodes by F value. The lowest on the back.
264         // If two nodes have same F value, the one with the lowest H value

```

jun 27, 17 14:46

compass.cpp

Page 5/11

```

265         // will be closer to the back.
266         std::vector<Node *>::iterator it = open_nodes.begin();
267         while ((!inserted) ^ (it != open_nodes.end())) {
268             if (((*it)->getFValue()) < new_node->getFValue()) {
269                 open_nodes.insert(it, new_node);
270                 inserted = true;
271             } else if (((*it)->getFValue() == new_node->getFValue()) {
272                 if (((*it)->getHvalue()) < new_node->getHvalue()) {
273                     open_nodes.insert(it, new_node);
274                     inserted = true;
275                 }
276             }
277             ++it;
278         }
279         if (!inserted) {
280             open_nodes.push_back(new_node);
281         }
282     }
283     if (!finished)
284         this->checkIfIsDestinyNeighbor(new_node, STEP);
285 }
286
287 void Compass::getRoad(Position& from, Node *destiny) {
288     road.push_back(destiny->getPosition());
289     Node* next_node = destiny->getParent();
290
291     Position current_pos = next_node->getPosition();
292     while ((current_pos.getX() != from.getX()) ^
293         (current_pos.getY() != from.getY())) {
294         this->addPositions(current_pos);
295         // road.push_back(current_pos);
296         next_node = next_node->getParent();
297         current_pos = next_node->getPosition();
298     }
299 }
300
301 Node *Compass::searchForClosestNode() {
302     Node* closest = closed_nodes.front();
303     for (auto x: closed_nodes) {
304         if ((x->getHvalue() < closest->getHvalue()) ^
305             ((x->getHvalue() == closest->getGvalue()) ^
306                 (x->getFvalue() < closest->getFvalue()))) {
307             closest = x;
308         }
309     }
310     return closest;
311 }
312
313 int Compass::getModuleOfSubtraction(int x, int y) {
314     if ((x - y) > 0)
315         return x - y;
316     return y - x;
317 }
318
319 void Compass::checkIfIsDestinyNeighbor(Node *node, int step) {
320     if ((node->getHvalue() ≤ HMIN*2) ^ (node->getHvalue() ≠ 0)) {
321         // get limits
322         int x_min = node->getPosition().getX() - step;
323         int x_max = node->getPosition().getX() + step;
324         int y_min = node->getPosition().getY() - step;
325         int y_max = node->getPosition().getY() + step;
326
327         Node *adj;
328         bool adj_new_g;
329         for (int x_pos = x_min; x_pos ≤ x_max; x_pos += step) {
330             for (int y_pos = y_min; y_pos ≤ y_max; y_pos += step) {

```

jun 27, 17 14:46

compass.cpp

Page 6/11

```

331         if (map.doesThisPositionExist(x_pos, y_pos)) {
332             adj = astar_map[x_pos][y_pos];
333             Size size = adj->getSize();
334             this->setHValueOnNode(adj);
335             if (adj->getHvalue() == 0) {
336                 // G value differs when the node is diagonal
337                 // or next to it
338                 if ((map.canIWalkToThisPosition(size, unit_id)) ^
339                     this->isNotMe(node, adj)) {
340                     if (this->isThisNodeOnDiagonal(node, adj)) {
341                         adj_new_g = this->writeGandSetParent(node, adj,
342                                                                 DIAGONALWAL
343                                                                 0);
344                     } else {
345                         adj_new_g = this->writeGandSetParent(node, adj,
346                                                                 SIDEWALK,
347                                                                 0);
348                     }
349                     if (adj_new_g)
350                         this->addToOpenInOrder(adj);
351                 }
352             }
353         }
354     }
355 }
356
357 bool Compass::canIWalkToThisPosition(Size &size) {
358     return map.canIWalkToThisPosition(size, unit_id);
359 }
360
361 double Compass::getTerrainFactorOn(int x, int y) {
362     return map.getTerrainFactorOn(x,y);
363 }
364
365 bool Compass::canBulletWalkToThisPosition(Size &size, Occupant &shooter,
366                                           Occupant &target) {
367     return map.canBulletWalkToThisPosition(size,shooter,target);
368 }
369
370 void Compass::changeUnitSize(Size &new_size) {
371     this->unit_size = new_size;
372 }
373
374 void Compass::changeUnitSpeed(int speed) {
375     this->unit_speed = speed;
376 }
377
378 Position Compass::getAValidPositionForDestiny(Position &destiny) {
379     Node *dest = astar_map[destiny.getX()][destiny.getY()];
380     Size size = dest->getSize();
381     if (map.canIWalkToThisPosition(size, unit_id)) {
382         return destiny;
383     } else {
384         return getClosestValidPosition(destiny);
385     }
386 }
387
388 Position Compass::getClosestValidPosition(Position &pos) {
389     bool found = false;
390     int i = 1;
391     Node* closest_node = astar_map[pos.getX()][pos.getY()];
392     while (!found) {
393         int x_min = pos.getX() - i;

```

jun 27, 17 14:46

compass.cpp

Page 7/11

```

396     int x_max = pos.getX() + i;
397     int y_min = pos.getY() - i;
398     int y_max = pos.getY() + i;
399
400     for (int x_pos = x_min; x_pos ≤ x_max; ++x_pos) {
401         if (map.doesThisPositionExist(x_pos, y_max)) {
402             Node *tmp = astar_map[x_pos][y_max];
403             Size size = tmp->getSize();
404             std::string terrain_type = map.getTerrainType(x_pos,y_max);
405             // if you fit on the position. When it's a vehicule check
406             // if it's different to water.
407             if ((map.canIWalkToThisPosition(size, unit_id)) ^
408                 (¬(unit_speed ≠ 2 ^ terrain_type == "Agua" ^
409                  ¬map.thereIsABridge(size)))) {
410                 found = true;
411                 closest_node = tmp;
412                 break;
413             }
414         }
415     }
416
417     if (!found) {
418         for (int x_pos = x_min; x_pos ≤ x_max; ++x_pos) {
419             if (map.doesThisPositionExist(x_pos, y_min)) {
420                 Node *tmp = astar_map[x_pos][y_min];
421                 Size size = tmp->getSize();
422                 std::string terrain_type = map.getTerrainType(x_pos,y_min);
423                 // if you fit on the position. When it's a vehicule check
424                 // if it's different to water.
425                 if ((map.canIWalkToThisPosition(size,unit_id)) ^
426                     (¬(unit_speed ≠ 2 ^ terrain_type == "Agua" ^
427                      ¬map.thereIsABridge(size)))) {
428                     found = true;
429                     closest_node = tmp;
430                     break;
431                 }
432             }
433         }
434     }
435
436     if (!found) {
437         for (int y_pos = y_min; y_pos ≤ y_max; ++y_pos) {
438             if (map.doesThisPositionExist(x_max, y_pos)) {
439                 Node *tmp = astar_map[x_max][y_pos];
440                 Size size = tmp->getSize();
441                 std::string terrain_type = map.getTerrainType(x_max,y_pos);
442                 // if you fit on the position. When it's a vehicule check
443                 // if it's different to water.
444                 if ((map.canIWalkToThisPosition(size, unit_id)) ^
445                     (¬(unit_speed ≠ 2 ^ terrain_type == "Agua" ^
446                      ¬map.thereIsABridge(size)))) {
447                     found = true;
448                     closest_node = tmp;
449                     break;
450                 }
451             }
452         }
453     }
454
455     if (!found) {
456         for (int y_pos = y_min; y_pos ≤ y_max; ++y_pos) {
457             if (map.doesThisPositionExist(x_min, y_pos)) {
458                 Node *tmp = astar_map[x_min][y_pos];
459                 Size size = tmp->getSize();
460                 std::string terrain_type = map.getTerrainType(x_min,y_pos);
461                 // if you fit on the position. When it's a vehicule check

```

jun 27, 17 14:46

compass.cpp

Page 8/11

```

462         // if it's different to water.
463         if ((map.canIWalkToThisPosition(size, unit_id)) ^
464             (¬(unit_speed ≠ 2 ^ terrain_type ≡ "Agua" ^
465              ¬map.thereIsABridge(size)))) {
466             found = true;
467             closest_node = tmp;
468             break;
469         }
470     }
471 }
472 }
473 }
474 ++i;
475 }
476 return closest_node→getSize().getPosition();
477 }
478
479 void Compass::changeUnitId(int id) {
480     this→unit_id = id;
481 }
482
483 void Compass::addPositions(Position& next_pos) {
484     Position pos = road.back();
485     bool increase_x = false, increase_y = false;
486     int x_max = 0, x_min = 0, y_max = 0, y_min = 0;
487     if (next_pos.getX() > pos.getX()) {
488         x_max = next_pos.getX();
489         x_min = pos.getX();
490         increase_x = true;
491     } else if (next_pos.getX() < pos.getX()) {
492         x_max = pos.getX();
493         x_min = next_pos.getX();
494     } else if (next_pos.getX() ≡ pos.getX()) {
495         x_max = pos.getX();
496         x_min = x_max;
497         increase_x = true;
498     }
499
500     if (next_pos.getY() > pos.getY()) {
501         y_max = next_pos.getY();
502         y_min = pos.getY();
503         increase_y = true;
504     } else if (next_pos.getY() < pos.getY()) {
505         y_max = pos.getY();
506         y_min = next_pos.getY();
507     } else if (next_pos.getY() ≡ pos.getY()) {
508         increase_y = true;
509         y_max = pos.getY();
510         y_min = y_max;
511     }
512
513     addPositionsInOrder(increase_x, increase_y, x_max, x_min, y_max, y_min);
514     Position last = road.back();
515     if (last.getX() ≠ next_pos.getX() ∨ last.getY() ≠ next_pos.getY())
516         road.push_back(next_pos);
517 }
518
519 void Compass::manageSteps(int &step, Position &start, Position &current_pos,
520                           Position &to) {
521     int tmp_h = HMIN * (this→getModuleOfSubtraction(current_pos.getX(),
522 to.getX()) + this→getModuleOfSubtraction(current_pos.getY(), to.getY()));
523     int closer_h = HMIN * CLOSERAREA * 2;
524     //Get smaller H depending on where start and destiny are
525     int close_x = 0, close_y = 0, mid_x = 0, mid_y = 0;
526     if (start.getX() ≤ to.getX()) {
527         close_x = this→getModuleOfSubtraction

```

jun 27, 17 14:46

compass.cpp

Page 9/11

```

528         (start.getX() + CLOSERAREA, to.getX());
529         mid_x = this→getModuleOfSubtraction
530             (start.getX() + MIDDLEAREA, to.getX());
531     } else if (start.getX() > to.getX()) {
532         close_x = this→getModuleOfSubtraction
533             (start.getX() - CLOSERAREA, to.getX());
534         mid_x = this→getModuleOfSubtraction
535             (start.getX() - MIDDLEAREA, to.getX());
536     }
537     if (start.getY() ≤ to.getY()) {
538         mid_y = this→getModuleOfSubtraction
539             (start.getY() + MIDDLEAREA, to.getY());
540         close_y = this→getModuleOfSubtraction
541             (start.getY() + CLOSERAREA, to.getY());
542     } else if (start.getY() > to.getY()) {
543         close_y = this→getModuleOfSubtraction
544             (start.getY() - CLOSERAREA, to.getY());
545         mid_y = this→getModuleOfSubtraction
546             (start.getY() - MIDDLEAREA, to.getY());
547     }
548     int start_h = HMIN * (close_x + close_y);
549     int mid_h = HMIN * (mid_x + mid_y);
550     // select step
551     if (tmp_h < closer_h ∨ getModule(start_h, closer_h) < tmp_h) {
552         step = 1;
553     } else if ((tmp_h > closer_h ∨ getModule(start_h, tmp_h) > closer_h)
554                ^ tmp_h < mid_h) {
555         if (unit_size.getWidth() > unit_size.getHeight()) {
556             step = (int) (unit_size.getHeight()*2);
557         } else {
558             step = (int) (unit_size.getWidth()*2);
559         }
560     } else {
561         step = (int) (unit_size.getHeight() * 10);
562     }
563 }
564
565 void Compass::setHValueOnNode(Node *node) {
566     Position tmp = node→getPosition();
567     int h_value = HMIN * (this→getModuleOfSubtraction(tmp.getX(),
568 destiny.getX()) + this→getModuleOfSubtraction(tmp.getY(), destiny.getY()));
569     node→setHValue(h_value);
570 }
571
572 void Compass::clearCompass() {
573     if (¬clear) {
574         this→road.clear();
575         this→closed_nodes.clear();
576         this→open_nodes.clear();
577         for (auto x: astar_map) {
578             for (auto y: x) {
579                 y→clean();
580             }
581         }
582         clear = true;
583     }
584 }
585
586 void Compass::addPositionsInOrder(bool increase_x, bool increase_y, int x_max,
587                                   int x_min, int y_max, int y_min) {
588     int i = x_min;
589     int j = y_min;
590     if (increase_x ^ increase_y) {
591         while (i < x_max ∨ j < y_max) {
592             if (i < x_max)
593                 ++i;

```

jun 27, 17 14:46

compass.cpp

Page 10/11

```

594         if (j < y_max)
595             ++j;
596         road.push_back(Position(i, j));
597     }
598     } else if (increase_x ^ !increase_y) {
599         j = y_max;
600         while (i < x_max ^ j > y_min) {
601             if (i < x_max)
602                 ++i;
603             if (j > y_min)
604                 --j;
605             road.push_back(Position(i, j));
606         }
607     } else if (!increase_x ^ increase_y) {
608         i = x_max;
609         while (i > x_min ^ j < y_max) {
610             if (i > x_min)
611                 --i;
612             if (j < y_max)
613                 ++j;
614             road.push_back(Position(i, j));
615         }
616     } else {
617         i = x_max;
618         j = y_max;
619         while (i > x_min ^ j > y_min) {
620             if (i > x_min)
621                 --i;
622             if (j > y_min)
623                 --j;
624             road.push_back(Position(i, j));
625         }
626     }
627 }
628
629 int Compass::getModule(int x, int y) {
630     if (x - y > 0) {
631         return x - y;
632     } else {
633         return y - x;
634     }
635 }
636
637 Occupant* Compass::checkForEnemiesOnRange(Occupant& unit, Size &range) {
638     return map.checkForEnemiesOn(range, unit);
639 }
640
641 bool Compass::checkIfItIsGrabbable(std::string& type) const {
642     return map.tellIfItIsGrabbable(type);
643 }
644
645 Compass::~Compass() {
646     if (!astar_map.empty()) {
647         int j = 0;
648         for (auto x: astar_map) {
649             int i = 0;
650             for (auto& y: x) {
651                 delete (y);
652                 ++i;
653             }
654             ++j;
655         }
656     }
657 }
658
659 bool Compass::checkIfItIsABuilding(std::string &type) const {

```

jun 27, 17 14:46

compass.cpp

Page 11/11

```

660     return map.tellIfItIsBuilding(type);
661 }

```

jun 27, 17 14:46

commandMonitor.h

Page 1/1

```

1  //
2  // Created by rodian on 15/06/17.
3  //
4
5  #ifndef Z_TPGRUPAL_COMMANDMONITOR_H
6  #define Z_TPGRUPAL_COMMANDMONITOR_H
7
8
9  #include "command.h"
10 class Command;
11 class ControlUnit;
12
13 class CommandMonitor {
14 private:
15     std::vector<Command> commands;
16     std::mutex& m;
17 public:
18     CommandMonitor(std::mutex& m);
19
20     void addCommand(std::string id, std::string& cmd, ControlUnit* control);
21
22     void copyCommands(std::vector<Command>& commands_copy);
23 };
24
25
26 #endif //Z_TPGRUPAL_COMMANDMONITOR_H

```

jun 27, 17 14:46

commandMonitor.cpp

Page 1/1

```

1  //
2  // Created by rodian on 15/06/17.
3  //
4
5  #include "commandMonitor.h"
6
7  CommandMonitor::CommandMonitor(std::mutex &m) : m(m) {}
8
9  void CommandMonitor::addCommand(std::string id, std::string& cmd,
10                                  ControlUnit* control) {
11      Lock l(m);
12      commands.push_back(Command(id, cmd, control));
13  }
14
15  void CommandMonitor::copyCommands(std::vector<Command> &commands_copy) {
16      Lock l(m);
17      for (auto cmd: commands) {
18          commands_copy.push_back(cmd);
19      }
20
21      commands.clear();
22  }

```

jun 27, 17 14:46

command.h

Page 1/1

```

1 //
2 // Created by rodian on 27/05/17.
3 //
4
5 #ifndef Z_TPGRUPAL_COMMAND_H
6 #define Z_TPGRUPAL_COMMAND_H
7
8 #include "controlUnit.h"
9 class ControlUnit;
10
11 class Command {
12 protected:
13     std::string player_id, cmd;
14     ControlUnit* control;
15     int unit_id, other_id, x,y;
16 public:
17     Command(std::string id, std::string& cmd, ControlUnit* control);
18
19     void run();
20
21     void operator()();
22
23 private:
24     std::string getNextData(std::string& line);
25
26     void analizeFactoryCommand(std::string& cmd, int id_factory );
27 };
28
29
30 #endif //Z_TPGRUPAL_COMMAND_H

```

jun 27, 17 14:46

command.cpp

Page 1/1

```

1 //
2 // Created by rodian on 27/05/17.
3 //
4
5 #include "command.h"
6
7 Command::Command(std::string id, std::string& cmd, ControlUnit* control) :
8     player_id(id), cmd(cmd), control(control) {}
9
10 void Command::run() {
11     std::string action = getNextData(cmd);
12     // if command is to move
13     if (action == "mv") {
14         // get id, then position (x,y)
15         std::string id_str = getNextData(cmd);
16         int id = std::stoi(id_str);
17         std::string x_str = getNextData(cmd);
18         int x = std::stoi(x_str);
19         std::string y_str = getNextData(cmd);
20         int y = std::stoi(y_str);
21         // make move
22         control->cmdMoveUnit(player_id, id, x, y);
23     } else if (action == "atk") {
24         std::string id_unit_str = getNextData(cmd);
25         int id_unit = std::stoi(id_unit_str);
26         std::string target_str = getNextData(cmd);
27         int target = std::stoi(target_str);
28         control->cmdAttack(player_id, id_unit, target);
29     } else if (action == "grab") {
30         std::string id_unit_str = getNextData(cmd);
31         int id_unit = std::stoi(id_unit_str);
32         std::string target_str = getNextData(cmd);
33         int target = std::stoi(target_str);
34         control->cmdGrab(player_id, id_unit, target);
35     } else if (action == "factory") {
36         std::string id_unit_str = getNextData(cmd);
37         int id_factory = std::stoi(id_unit_str);
38         std::string factory_cmd = getNextData(cmd);
39         analizeFactoryCommand(factory_cmd, id_factory);
40     }
41 }
42
43 void Command::operator()() {
44     this->run();
45 }
46
47 std::string Command::getNextData(std::string& line) {
48     std::size_t found = line.find('-');
49     std::string data = line.substr(0, found);
50     line.erase(0, found+1);
51     return data;
52 }
53
54 void Command::analizeFactoryCommand(std::string& cmd, int id_factory) {
55     std::string action = getNextData(cmd);
56     if (action == "create") {
57         control->cmdFactoryCreate(player_id, id_factory);
58     } else if (action == "prev") {
59         control->cmdFactoryPrev(player_id, id_factory);
60     } else if (action == "next") {
61         control->cmdFactoryNext(player_id, id_factory);
62     } else if (action == "current") {
63         control->cmdFactoryCurrent(player_id, id_factory);
64     }
65 }

```

jun 27, 17 14:46

cell.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_CELL_H
2  #define Z_TPGRUPAL_CELL_H
3
4  #include "Terrain.h"
5  #include "size.h"
6  #include <iostream>
7  #include <string>
8  #include <stdbool>
9  #include <mutex>
10
11 // Class Cell to represent a position on the map
12 class Cell {
13     private:
14         Terrain terrain;
15         Size size;
16
17     public:
18         // Cell constructor on position (x,y). Builds the Terrain inside it and
19         // always has an Occupant. The empty Cell will be the one who has an
20         // Occupant with id = -1.
21         Cell(int x, int y, int width, int lenght,
22             std::string& kind, int factor);
23
24         // Returns a string with the king of Terrain
25         std::string getTerrainType() const;
26
27         // Returns the movement factor of the Terrain
28         double getMovementFactor() const;
29
30         Position getPosition() const;
31
32         int getWidthOfCell();
33
34         bool areYouOnThisPosition(int x_pos, int y_pos);
35
36         bool isThereACollision(Size& size);
37
38         ~Cell();
39 };
40
41 #endif

```

jun 27, 17 14:46

cell.cpp

Page 1/1

```

1  //
2  // Created by rodian on 13/05/17.
3  //
4
5  #include "cell.h"
6
7  Cell::Cell(int x, int y, int width, int lenght, std::string &kind, int factor):
8      size(x, y, width, lenght), terrain(kind,factor) {}
9
10 std::string Cell::getTerrainType() const{
11     return this->terrain.getKind();
12 }
13
14 double Cell::getMovementFactor() const {
15     return this->terrain.getFactor();
16 }
17
18 Position Cell::getPosition() const {
19     return this->size.getPosition();
20 }
21
22 bool Cell::areYouOnThisPosition(int x_pos, int y_pos) {
23     return this->size.areYouOnThisPoint(x_pos, y_pos);
24 }
25
26 int Cell::getWidthOfCell() {
27     return size.getWidth();
28 }
29
30 bool Cell::isThereACollision(Size &size) {
31     return size.isThereACollision(size);
32 }
33
34 Cell::~Cell() {}
35
36

```

jun 27, 17 14:46	bullet.h	Page 1/1
------------------	----------	----------

```

1  //
2  // Created by rodian on 22/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_BULLET_H
6  #define Z_TPGRUPAL_BULLET_H
7
8  #include <string>
9  #include <vector>
10 #include "size.h"
11 #include "Occupant.h"
12
13 class Bullet {
14 private:
15     std::string type;
16     int damage, w_speed;
17     Size w_size;
18     bool hit, must_disappear;
19     Occupant* target;
20     std::vector<Position> road;
21     int id;
22
23 public:
24     Bullet(std::string type, int damage, int w_speed, Size& w_size,
25           Occupant* target);
26
27     Bullet(std::string type, int damage, int w_speed, Size& w_size);
28
29     // Pursues the Target in straight line on each TIC til hits
30     void shotTarget(Occupant* target);
31
32     void calculateRoadToTarget();
33
34     void move();
35
36     bool didHit();
37
38     Position calculateNextPosition(double a, double b, int x);
39
40     Position calculateNextInvertPosition(double a, double b, int y);
41
42     Size getSize() const;
43
44     bool isRoadEmpty();
45
46     std::vector<Position>& getRoad();
47
48     void damageThis(Occupant* occupant);
49
50     void setStartLocation(int x, int y);
51
52     void setCorrectId(int id);
53
54     int getId() const;
55
56     Position getPosition() const;
57
58     void disappear();
59
60     bool doYouHaveToDisappear();
61
62     std::string getType();
63 };
64
65 #endif //Z_TPGRUPAL_BULLET_H

```

jun 27, 17 14:46	bullet.cpp	Page 1/3
------------------	------------	----------

```

1  //
2  // Created by rodian on 22/05/17.
3  //
4
5  #include "bullet.h"
6
7  Bullet::Bullet(std::string type, int dmg, int w_speed, Size& w_size,
8                Occupant* target) : type(type), damage(dmg), w_speed(w_speed),
9                                   w_size(w_size), hit(false), must_disappear(false)
10                                   , target(target), id(0) {}
11
12  Bullet::Bullet(std::string type, int damage, int w_speed, Size& w_size) :
13                type(type), damage(damage), w_speed(w_speed),
14                w_size(w_size), hit(false), must_disappear(false), id(0) {}
15
16
17  void Bullet::shotTarget(Occupant* target) {
18      this->target = target;
19      calculateRoadToTarget();
20  }
21
22  bool Bullet::didHit() {
23      return hit;
24  }
25
26  void Bullet::calculateRoadToTarget() {
27      road.clear();
28      Position target_pos = target->getPosition();
29      Position bullet_pos = w_size.getPosition();
30
31      if (bullet_pos.getX() - target_pos.getX() != 0) {
32          // solve the equation system getting the linear function y = ax + b
33          int res_y = (bullet_pos.getY() - target_pos.getY());
34          int res_x = (bullet_pos.getX() - target_pos.getX());
35          double a = (double) res_y / res_x;
36          double b = (double) (bullet_pos.getY() - (a * bullet_pos.getX()));
37
38          if (bullet_pos.getX() > target_pos.getX()) {
39              for (int i = target_pos.getX(); i < bullet_pos.getX(); ++i) {
40                  road.push_back(calculateNextPosition(a, b, i));
41              }
42          } else if (bullet_pos.getX() < target_pos.getX()) {
43              for (int i = target_pos.getX(); i > bullet_pos.getX(); --i) {
44                  road.push_back(calculateNextPosition(a, b, i));
45              }
46          }
47      } else {
48          // solve the equation system getting the linear function x = ay + b
49          double a = (bullet_pos.getX() - target_pos.getX()) /
50                    (bullet_pos.getY() - target_pos.getY());
51          double b = bullet_pos.getX() - (a * bullet_pos.getY());
52
53          if (bullet_pos.getY() > target_pos.getY()) {
54              for (int i = target_pos.getY(); i < bullet_pos.getY(); ++i) {
55                  road.push_back(calculateNextInvertPosition(a, b, i));
56              }
57          } else if (bullet_pos.getY() < target_pos.getY()) {
58              for (int i = target_pos.getY(); i > bullet_pos.getY(); --i) {
59                  road.push_back(calculateNextInvertPosition(a, b, i));
60              }
61          }
62      }
63  }
64
65  Position Bullet::calculateNextPosition(double a, double b, int x) {
66      double temp_y = a * x + b;

```



jun 27, 17 14:46

bullet.cpp

Page 2/3

```

67     return Position(x, (int)temp_y);
68 }
69
70 Size Bullet::getSize() const {
71     return this->w_size;
72 }
73
74 void Bullet::move() {
75     if (!hit) {
76         // cause target might be moving, recalculate road and then move
77         calculateRoadToTarget();
78         int distance = w_speed;
79         int steps = 0;
80         while (!road.empty() ^ steps <= distance) {
81             Position pos = road.back();
82             this->w_size.moveTo(pos.getX(), pos.getY());
83             road.pop_back();
84             ++steps;
85         }
86         // If you get the target, inflict damage
87         Size targ_size = target->getSize();
88         if (this->w_size.isThereACollision(targ_size)) {
89             target->reduceLifeBy(damage);
90             hit = true;
91         }
92     }
93 }
94
95 Position Bullet::calculateNextInvertPosition(double a, double b, int y) {
96     double temp_x = a * y + b;
97     return Position((int) temp_x, y);
98 }
99
100 bool Bullet::isRoadEmpty() {
101     return road.empty();
102 }
103
104 std::vector<Position>& Bullet::getRoad() {
105     return road;
106 }
107
108 void Bullet::damageThis(Occupant* other_target) {
109     other_target->reduceLifeBy(damage);
110     hit = true;
111 }
112
113 void Bullet::setStartLocation(int x, int y) {
114     this->w_size.moveTo(x,y);
115 }
116
117 void Bullet::setCorrectId(int id) {
118     this->id = id;
119 }
120
121 int Bullet::getId() const {
122     return id;
123 }
124
125 Position Bullet::getPosition() const {
126     return this->w_size.getPosition();
127 }
128
129 bool Bullet::doYouHaveToDisapear() {
130     return must_disapear;
131 }
132

```

jun 27, 17 14:46

bullet.cpp

Page 3/3

```

133 void Bullet::disapear() {
134     must_disapear = true;
135 }
136
137 std::string Bullet::getType() {
138     return this->ttype;
139 }
140

```

jun 27, 17 14:46

## Random.h

Page 1/1

```
1 #ifndef Z_TPGRUPAL_RANDOM_H
2 #define Z_TPGRUPAL_RANDOM_H
3
4 #include <ctime>
5
6 /* Random number generator */
7 class Random {
8     unsigned int seed;
9 public:
10     Random();
11
12     /* Returns a random int from 0 to INT_MAX */
13     int generate();
14 };
15
16
17 #endif //Z_TPGRUPAL_RANDOM_H
```

jun 27, 17 14:46

## Random.cpp

Page 1/1

```
1 #include <stdlib.h>
2 #include "Random.h"
3
4 Random::Random() : seed((unsigned int) time(NULL)) {
5 }
6
7 int Random::generate() {
8     return rand_r(&seed);
9 }
```

jun 27, 17 14:46

## MapGenerator.h

Page 1/2

```

1  #ifndef Z_TPGRUPAL_MAPGENERATOR_H
2  #define Z_TPGRUPAL_MAPGENERATOR_H
3
4  #include <fstream>
5  #include <vector>
6  #include <string>
7  #include <pugixml.hpp>
8  #include "Random.h"
9
10 // Config variables, percentages / amounts of features the map will have
11 #define ROCK_PCT 2
12 #define BRIDGE_AMT size / 20
13 #define RIVER_END_PCT 5
14 #define FORTS_AMT 4
15
16
17
18 /* Map generator. Randomly generates a readable .xml map file basing off the
19  * passed arguments on the constructor. The maps are saved to the 'maps' folder
20  * in the root directory. */
21 class MapGenerator {
22     std::vector<std::vector<bool>> liquid_cells;
23     std::ofstream output;
24     int size;
25     float lava_pct;
26     float water_pct;
27     int water_cells;
28     int lava_cells;
29     int terr;
30
31     std::string name;
32     // Random number generator
33     Random r;
34
35 public:
36     MapGenerator(int size, float lava_pct,
37                 float water_pct, int territories);
38     ~MapGenerator();
39
40     // Generate the map, saving it to "maps/<name>.xml".
41     void generate(const std::string& name);
42
43 private:
44     /* Inits a map */
45     void generate_blank_map(pugi::xml_node root_node);
46
47     /* Generates cell_amt of cells, ordered in a river like structure, in the
48      * map given by the root node. The cells are written with a "terrain"
49      * attribute as children of the root node, with the value given by the
50      * string 'terrain'. */
51     std::vector<std::vector<bool>> generate_rivers(pugi::xml_node root_node,
52                                                    int cell_amt,
53                                                    const std::string &terrain);
54
55     /* Generates river-like paths in a 'size' big square map, represented by
56      * a matrix of boolean values. */
57     void generate_path(int amt, std::vector<std::vector<bool>>& path);
58
59     /* Generates rocks */
60     void generate_rocks(pugi::xml_node root);
61
62     /* Generates FORTS_AMT forts in the map, placed separate from each other */
63     void generate_territories(pugi::xml_node root);
64
65     /* Generates 1 or 2 factories inside the territory delimited by the
66      * specified bounds */

```

jun 27, 17 14:46

## MapGenerator.h

Page 2/2

```

67     void generate_factories(pugi::xml_node &territory, int min_x, int min_y, int
68                             max_x,
69                             int max_y, pugi::xml_node &map);
70     };
71
72 #endif //Z_TPGRUPAL_MAPGENERATOR_H

```

jun 27, 17 14:46

## MapGenerator.cpp

Page 1/5

```

1  #include <fstream>
2  #include <string>
3  #include <vector>
4  #include <iostream>
5  #include "MapGenerator.h"
6  #include <pugixml.hpp>
7  #include <random>
8
9  #define UNIT 0
10 #define VEHICLE 1
11
12 #define TERRAIN "terrain"
13
14 MapGenerator::MapGenerator(int size, float lava_pct,
15                             float water_pct, int territories) :
16     size(size),
17     lava_pct(lava_pct),
18     water_pct(water_pct),
19     terr(territories)
20 {
21     for (int i = 0; i < size; ++i) {
22         std::vector<bool> row;
23         for (int j = 0; j < size; ++j) {
24             row.push_back(false);
25         }
26         liquid_cells.push_back(row);
27     }
28     water_cells = (int) (size * size * water_pct / 100);
29     lava_cells = (int) (size * size * lava_pct / 100);
30
31     /* Adjustment to size to split territories evenly */
32     int territories_per_row = (int) floor(sqrt(terr));
33     if (size % territories_per_row) {
34         this->size = size - territories_per_row;
35     }
36 }
37
38 void MapGenerator::generate_blank_map(pugi::xml_node root_node) {
39     for (int i = 0; i < size; ++i) {
40         pugi::xml_node row = root_node.append_child("Row");
41         for (int j = 0; j < size; ++j) {
42             pugi::xml_node cell = row.append_child("Cell");
43             pugi::xml_attribute attr = cell.append_attribute(TERRAIN);
44             attr.set_value("Tierra");
45         }
46     }
47 }
48
49
50
51 std::vector<std::vector<bool>>
52 MapGenerator::generate_rivers(pugi::xml_node root_node, int cell_amt,
53                               const std::string &terrain) {
54     std::vector<std::vector<bool>> map;
55     generate_path(cell_amt, map);
56     int count_y = 0;
57     for (pugi::xml_node& row : root_node.children()) {
58         int count_x = 0;
59         for (pugi::xml_node& node : row.children()) {
60             if (map[count_x][count_y]) {
61                 node.attribute(TERRAIN).set_value(terrain.c_str());
62             }
63             count_x++;
64         }
65         count_y++;
66     }

```

jun 27, 17 14:46

## MapGenerator.cpp

Page 2/5

```

67
68     for (int i = 0; i < size; i++) {
69         for (int j = 0; j < size; ++j) {
70             if (map[i][j]) {
71                 liquid_cells[i][j] = true;
72             }
73         }
74     }
75     return map;
76 }
77
78
79 void MapGenerator::generate_path(int amt,
80                                  std::vector<std::vector<bool>>& path) {
81     for (int i = 0; i < size; ++i) {
82         std::vector<bool> row;
83         for (int j = 0; j < size; ++j) {
84             row.push_back(false);
85         }
86         path.push_back(row);
87     }
88
89     int river_x = r.generate() % size;
90     int river_y = r.generate() % size;
91
92     while (amt) {
93         path[river_x][river_y] = true;
94
95         bool found = false;
96
97         while (!found) {
98             int end = r.generate() % 100;
99             if (end < RIVER_END_PCT) { // Start another river somewhere else
100                 river_x = r.generate() % size;
101                 river_y = r.generate() % size;
102             }
103             // Grab an adjacent tile randomly to be the next water tile
104             int next = r.generate() % 4;
105             int next_x, next_y;
106             if (next == 0) {
107                 next_x = 1;
108                 next_y = 0;
109             } else if (next == 1) {
110                 next_x = 0;
111                 next_y = -1;
112             } else if (next == 2) {
113                 next_x = -1;
114                 next_y = 0;
115             } else {
116                 next_x = 0;
117                 next_y = 1;
118             }
119             next_x += river_x;
120             next_y += river_y;
121
122
123             // Check for out of bounds
124             if (!(next_x > 0 ^ next_y > 0 ^ next_x < size ^ next_y < size)) {
125                 continue;
126             }
127
128             if (!path[next_x][next_y]) {
129                 found = true;
130                 amt--;
131                 river_x = next_x;
132                 river_y = next_y;

```

jun 27, 17 14:46

## MapGenerator.cpp

Page 3/5

```

133     }
134 }
135 }
136 }
137
138 void MapGenerator::generate_rocks(pugi::xml_node root) {
139     root.set_name("Structures");
140     for (int i = 0; i < size; ++i) {
141         for (int j = 0; j < size; ++j) {
142             if (!liquid_cells[i][j]) {
143                 int chance = r.generate() % 100;
144                 if (chance < ROCK_PCT) {
145                     pugi::xml_node rock = root.append_child("Struct");
146                     rock.append_attribute("Type").set_value("Rock");
147                     rock.append_attribute("x").set_value(i);
148                     rock.append_attribute("y").set_value(j);
149                 }
150             }
151         }
152     }
153 }
154 }
155
156 void MapGenerator::generate(const std::string& name) {
157     std::string path = "maps/" + name + ".xml";
158     pugi::xml_document document;
159     pugi::xml_node root = document.append_child("Map");
160     pugi::xml_node terrain = root.append_child("Terrain");
161     generate_blank_map(terrain);
162
163     generate_territories(root);
164     generate_rivers(terrain, water_cells, "Agua");
165     generate_rivers(terrain, lava_cells, "Lava");
166
167     pugi::xml_node structs = root.append_child("Structs");
168     generate_rocks(structs);
169     bool saved = document.save_file(path.c_str());
170     if (!saved) {
171         std::cout << "Error saving map to " << path << std::endl;
172     }
173 }
174
175 void MapGenerator::generate_territories(pugi::xml_node root) {
176     pugi::xml_node forts = root.append_child("Territories");
177
178     /* Choose exactly FORTS_AMT of territories to be designed as central.
179      * There's one fort for each expected player in the map */
180     int fort_territories[FORTS_AMT];
181     for (int k = 0; k < FORTS_AMT; ++k) {
182         bool found = false;
183         while (!found) {
184             int position = r.generate() % terr;
185             bool repeat = false;
186             for (int i = 0; i < k; ++i) {
187                 if (fort_territories[i] == position) {
188                     repeat = true;
189                 }
190             }
191             if (repeat) {
192                 continue;
193             }
194             fort_territories[k] = position;
195             found = true;
196         }
197     }
198 }

```

jun 27, 17 14:46

## MapGenerator.cpp

Page 4/5

```

199     }
200 }
201
202 double size_sqrt = sqrt(terr);
203 int territories_x = (int) floor(size_sqrt);
204 int territories_y = (int) ceil(size_sqrt);
205
206 int div_x = size / territories_x;
207 int div_y = size / territories_y;
208 int count = 0;
209 for (int i = 0; i < territories_y; ++i) {
210     for (int j = 0; j < territories_x; ++j) {
211         /* Randomize positions in the territories */
212         int terr_min_x = div_x * j,
213             terr_min_y = div_y * i,
214             terr_max_x = div_x * (j + 1) - 1,
215             terr_max_y = div_y * (i + 1) - 1;
216
217         std::string name = "Flag";
218         for (int k = 0; k < FORTS_AMT; ++k) {
219             if (fort_territories[k] == count) {
220                 name = "Fort";
221             }
222         }
223
224         bool found = false;
225         int flag_x = 0;
226         int flag_y = 0;
227
228         while (!found) {
229             flag_x = terr_min_x + r.generate() % (size / terr);
230             flag_y = terr_min_y + r.generate() % (size / terr);
231             if (!liquid_cells[flag_x][flag_y]) {
232                 found = true;
233             }
234         }
235         pugi::xml_node flag = forts.append_child(name.c_str());
236         flag.append_attribute("center_x").set_value(flag_x);
237         flag.append_attribute("center_y").set_value(flag_y);
238         flag.append_attribute("min_x").set_value(terr_min_x);
239         flag.append_attribute("min_y").set_value(terr_min_y);
240         flag.append_attribute("max_x").set_value(terr_max_x);
241         flag.append_attribute("max_y").set_value(terr_max_y);
242
243         pugi::xml_node map = root.child("Terrain");
244         generate_factories(flag, terr_min_x, terr_min_y, terr_max_x,
245                             terr_max_y, map);
246         count++;
247     }
248 }
249
250 void MapGenerator::generate_factories(pugi::xml_node &territory, int min_x,
251                                       int min_y, int max_x,
252                                       int max_y, pugi::xml_node &map) {
253     int territories = 2;
254     for (int i = 0; i < territories; ++i) {
255         bool found = false;
256         while (!found) {
257             /* Randomize the position, inside the territory */
258             int fact_x = r.generate() % (max_x - min_x) + min_x;
259             int fact_y = r.generate() % (max_y - min_y) + min_y;
260
261             /* Select type: unit or vehicle */
262             int unit_or_vehicle_factory = r.generate() % 2;
263             std::string type;
264         }
265     }

```

jun 27, 17 14:46

## MapGenerator.cpp

Page 5/5

```

265     if (unit_or_vehicle_factory == UNIT) {
266         type = "UnitFactory";
267     } else if (unit_or_vehicle_factory == VEHICLE) {
268         type = "VehicleFactory";
269     }
270
271     int count_x = 0;
272     for (pugi::xml_node row : map.children()) {
273         int count_y = 0;
274         for (pugi::xml_node cell : row.children()) {
275             if (fact_x == count_x ^ fact_y == count_y) {
276                 const char* terrain = cell.attribute(TERRAIN).value();
277                 if (!liquid_cells[fact_x][fact_y]) {
278                     pugi::xml_node factory =
279                         territory.append_child(type.c_str());
280                     factory.append_attribute("x").set_value(fact_x);
281                     factory.append_attribute("y").set_value(fact_y);
282                     found = true;
283                     break;
284                 }
285             }
286             count_y++;
287         }
288         if (found) {
289             break;
290         }
291         count_x++;
292     }
293 }
294 }
295 }
296 }
297
298 MapGenerator::~MapGenerator() {
299     if (output.is_open()) {
300         output.close();
301     }
302 }

```

jun 27, 17 14:46

## main.cpp

Page 1/1

```

1  #include <iostream>
2  #include "MapGenerator.h"
3
4  #define SIZE 1
5  #define WATER_PCT 2
6  #define LAVA_PCT 3
7  #define TERRITORIES 4
8  #define NAME 5
9
10 int main(int argc, char **argv) {
11     if (argc < 6) {
12         std::cout << "Usage: " << argv[0] << " <size> <water_pct> <lava_pct> "
13             << "<territories_amt> <name>" << std::endl;
14         return 1;
15     }
16
17     try {
18         int size = std::stoi(argv[SIZE]),
19             water = std::stoi(argv[WATER_PCT]),
20             lava = std::stoi(argv[LAVA_PCT]),
21             territories = std::stoi(argv[TERRITORIES]);
22         std::string name = argv[NAME];
23         MapGenerator generator(size, lava, water, territories);
24         generator.generate(name);
25     } catch (const std::invalid_argument& e) {
26         std::cout << "Usage: " << argv[0] << " <size> <water_pct> <lava_pct> "
27             << "<territories_amt> <name>" << std::endl;
28         return 1;
29     }
30 }

```

jun 27, 17 14:46

**Thread.h**

Page 1/1

```
1  #ifndef Z_TPGRUPAL_THREAD_H
2  #define Z_TPGRUPAL_THREAD_H
3
4
5  #include <thread>
6
7  class Thread {
8  protected:
9      std::thread thread;
10
11 public:
12     void start();
13
14     virtual void run() = 0;
15
16     void join();
17 };
18
19
20 #endif //Z_TPGRUPAL_THREAD_H
```

jun 27, 17 14:46

**Thread.cpp**

Page 1/1

```
1  #include "Thread.h"
2
3  void Thread::start() {
4      thread = std::thread(&Thread::run, this);
5  }
6
7  void Thread::join() {
8      this->thread.join();
9  }
```

jun 27, 17 14:46

split.h

Page 1/1

```

1  #ifndef TP3TALLER_COMMON_SPLIT_H
2  #define TP3TALLER_COMMON_SPLIT_H
3
4  #include <vector>
5  #include <string>
6
7  /* Splits the input string in as many strings as possible, using the char
8   * 'delim' as the delimiter between the result strings. Returns the smaller
9   * strings in a vector. The function is guaranteed to return with at least
10  * one string in the vector.
11  */
12 namespace utils {
13     std::vector<std::string> split(const std::string& input, char delim);
14 }
15
16 #endif //TP3TALLER_COMMON_SPLIT_H
17

```

jun 27, 17 14:46

split.cpp

Page 1/1

```

1  #include <vector>
2  #include <string>
3  #include <sstream>
4
5  namespace utils {
6      std::vector<std::string> split(const std::string& input, char delim) {
7          std::vector<std::string> result;
8          std::istringstream stream(input);
9          for (std::string field; std::getline(stream, field, delim); ) {
10              result.push_back(field);
11          }
12          return result;
13      }
14  };
15

```



jun 27, 17 14:46

## socket.h

Page 1/1

```

1  #ifndef TP3TALLER_COMMON_SOCKET_H
2  #define TP3TALLER_COMMON_SOCKET_H
3
4  #include <string>
5  #include "socketError.h"
6
7  #define LISTEN_BACKLOG 10 // Amt. of connections to have in the accept backlog
8
9  // Socket class. Wraps functionality of glibc's socket functions.
10 class Socket {
11     int fd;
12
13 public:
14     /* Server constructor. Creates a socket, binds and listens to the specified
15      * port. */
16     explicit Socket(int port);
17
18     /* Client constructor. Creates a socket and attempts to connect to the
19      * specified address/port. Raises exception if the connection fails. */
20     Socket(const char *addr, int port);
21
22     ~Socket();
23
24     // Returns a new client.
25     Socket accept_client();
26
27     // Sends/recieves len bytes of data
28     ssize_t send(const char *msg, unsigned int len);
29     ssize_t receive(char *dest, size_t len);
30
31     // Wrapper for socket shutdown/close
32     void shutdown();
33     void close(); // Effectively makes the socket object useless
34
35     bool is_valid();
36     // Move constructor
37     Socket(Socket& other);
38
39 private:
40     Socket();
41
42     Socket(Socket&) = delete;
43     void operator=(Socket&) = delete;
44 };
45
46 #endif //TP3TALLER_COMMON_SOCKET_H

```

jun 27, 17 14:46

## socketError.h

Page 1/1

```

1  //
2  // Created by rodian on 22/05/17.
3  //
4
5  #ifndef Z_TPGRUPAL_SOCKETERROR_H
6  #define Z_TPGRUPAL_SOCKETERROR_H
7
8  #include <iostream>
9  #include <cstring>
10
11  //////////////////////////////////
12  // SocketError Class to warn
13  // of an error on the socket
14  //////////////////////////////////
15
16 class SocketError : public std::exception {
17 private:
18     char buffer[124];
19
20 public:
21     explicit SocketError(const char* message, ...) noexcept;
22
23     // Returns the error message
24     virtual const char* what() const noexcept;
25 };
26
27 #endif //Z_TPGRUPAL_SOCKETERROR_H

```

jun 27, 17 14:46

socketError.cpp

Page 1/1

```

1 //
2 // Created by rodian on 22/05/17.
3 //
4
5 #include "socketError.h"
6
7 SocketError::SocketError(const char *message, ...) noexcept {
8     strncpy(buffer, message, strlen(message));
9 }
10
11 const char* SocketError::what() const noexcept {
12     return buffer;
13 }

```

jun 27, 17 14:46

socket.cpp

Page 1/2

```

1 #include <sys/socket.h>
2 #include <unistd.h>
3 #include <netinet/in.h>
4 #include <arpa/inet.h>
5 #include <cstring>
6 #include <utility>
7 #include <iostream>
8 #include <string>
9 #include "socket.h"
10
11 Socket::Socket(int port) {
12     fd = socket(AF_INET, SOCK_STREAM, 0);
13     if (fd < 0) {
14         throw SocketError("Couldn't create a socket!\n");
15     }
16
17     struct sockaddr_in srv;
18     memset(&srv, 0, sizeof(srv));
19     srv.sin_family = AF_INET;
20     srv.sin_addr.s_addr = htonl(INADDR_ANY);
21     srv.sin_port = htons((uint16_t) port);
22
23     int yes = 1;
24     if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) < 0)
25         throw SocketError("setsockopt(SO_REUSEADDR) failed");
26
27     int error = bind(fd, (struct sockaddr *) &srv, sizeof(srv));
28     if (error) {
29         throw SocketError("Error binding socket on creation! "
30                             "Most likely port already in use");
31     }
32
33     listen(fd, LISTEN_BACKLOG);
34 }
35
36 Socket::Socket(const char *addr, int port) {
37     fd = socket(AF_INET, SOCK_STREAM, 0);
38     struct sockaddr_in srv;
39     srv.sin_family = AF_INET;
40     srv.sin_port = htons((uint16_t)port);
41     srv.sin_addr.s_addr = inet_addr(addr);
42
43     socklen_t len = (socklen_t)sizeof(struct sockaddr);
44     int error = connect(fd, (struct sockaddr *) &srv, len);
45
46     if (error) {
47         throw SocketError("Error connecting to server!");
48     }
49 }
50
51
52 Socket Socket::accept_client() {
53     struct sockaddr_in client;
54     socklen_t clilen = (socklen_t) sizeof(struct sockaddr_in);
55
56     int client_fd = accept(fd, (struct sockaddr *) &client, &clilen);
57     if (client_fd < 0 || fd < 0) {
58         throw SocketError("Socket disconnected");
59     }
60
61     Socket client_socket;
62     client_socket.fd = client_fd;
63     return client_socket;
64 }
65
66 Socket::Socket() {

```

jun 27, 17 14:46

socket.cpp

Page 2/2

```

67 }
68
69 Socket::Socket(Socket& other) {
70     fd = other.fd;
71     other.fd = -1; // "Deactivates" other
72 }
73
74 ssize_t Socket::send(const char *msg, unsigned int len) {
75     size_t total_bytes = 0;
76     ssize_t sent = 1;
77
78     // Sends msg until it's complete OR socket_send returns 0 (connection
79     // closed)
80     while (total_bytes < len ^ sent) {
81         sent = ::send(fd, msg + total_bytes, len - total_bytes,
82                     MSG_NOSIGNAL);
83         if (sent < 0) {
84             return -1;
85         }
86         total_bytes += sent;
87     }
88     return total_bytes;
89 }
90
91
92 ssize_t Socket::receive(char *dest, size_t len) {
93     ssize_t received = 1;
94     size_t total_bytes = 0;
95
96     // Writes to dest until it's complete OR socket_recv returns 0 (connection
97     // closed)
98     while (total_bytes < len ^ received) {
99         received = recv(fd, dest + total_bytes, len - total_bytes,
100                     MSG_NOSIGNAL);
101         if (received < 0) {
102             return -1;
103         }
104         total_bytes += received;
105     }
106     return total_bytes;
107 }
108
109
110 Socket::~Socket() {
111     if (fd > 0) {
112         close();
113     }
114 }
115
116 void Socket::shutdown() {
117     ::shutdown(fd, SHUT_RDWR);
118     fd = -1;
119 }
120
121 void Socket::close() {
122     ::close(fd);
123     fd = -1;
124 }
125
126 bool Socket::is_valid() {
127     return fd > 0;
128 }

```

jun 27, 17 14:46

messenger.h

Page 1/1

```

1 //
2 // Created by rodian on 22/05/17.
3 //
4
5 #ifndef Z_TPGRUPAL_MESSENGER_H
6 #define Z_TPGRUPAL_MESSENGER_H
7
8 #include "socket.h"
9 #include <iostream>
10 #include <string>
11 ///////////////////////////////////////////////////
12 // Messenger Class meant to use sockets
13 // to send messages between Client and
14 // Server using a specific protocol.
15 // Send lenght of message first, then the message.
16 ///////////////////////////////////////////////////
17 class Messenger{
18 private:
19     Socket socket;
20
21 public:
22     // Recieves a unique socket to send
23     // and recieves messages from
24     explicit Messenger(Socket& socket);
25
26     // Recieves a Message from the remote
27     // connected socket.
28     // Returns the message on a string
29     std::string recieveMessage();
30
31     // Sends a message to the remote socket
32     // Recieves the message on a string
33     void sendMessage(const std::string &message);
34
35     // Shuts down the socket for read and write
36     void shutdown();
37
38     // If the sockets are still connected returns true
39     // otherwise false.
40     bool isConnected();
41
42     ~Messenger();
43
44     Messenger(Messenger& other);
45 };
46
47
48 #endif //Z_TPGRUPAL_MESSENGER_H

```

jun 27, 17 14:46

messenger.cpp

Page 1/1

```

1  #include <netinet/in.h>
2  #include <string>
3  #include <stdint>
4  #include <stdint-gcc.h>
5  #include "socket.h"
6  #include "messenger.h"
7
8  Messenger::Messenger(Socket& socket) : socket(std::move(socket)) {}
9
10
11 std::string Messenger::recieveMessage() {
12     // Receive length first, then the message
13     uint32_t len = 0;
14     socket.receive((char*) &len, sizeof(len));
15     len = ntohl(len);
16     char* buf = new char[len];
17     ssize_t sent = socket.receive(buf, len);
18     if (sent <= 0) {
19         socket.close();
20         throw(SocketError("Socket closed"));
21     }
22     std::string result(buf);
23
24     delete[] buf;
25     return result;
26 }
27
28 void Messenger::sendMessage(const std::string &message) {
29     if (isConnected()) {
30         uint32_t len = (uint32_t) message.size() + 1;
31         // Send length first, then the message
32         uint32_t network_len = htonl(len);
33         socket.send((char *) &network_len, sizeof(network_len));
34         socket.send(message.c_str(), len);
35     }
36 }
37
38 void Messenger::shutdown() {
39     socket.shutdown();
40 }
41
42 bool Messenger::isConnected() {
43     return socket.is_valid();
44 }
45
46 Messenger::~Messenger() {
47 }
48
49 Messenger::Messenger(Messenger &other) : socket(std::move(other.socket)){
50 }

```

jun 27, 17 14:46

Lock.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_LOCK_H
2  #define Z_TPGRUPAL_LOCK_H
3
4
5  #include <mutex>
6
7  class Lock {
8  private:
9      std::mutex &m;
10
11 public:
12     explicit Lock(std::mutex &m);
13
14     ~Lock();
15 };
16
17
18 #endif //Z_TPGRUPAL_LOCK_H

```

jun 27, 17 14:46

**Lock.cpp**

Page 1/1

```

1  #include "Lock.h"
2
3  Lock::Lock(std::mutex &m) : m(m){
4      m.lock();
5  }
6
7  Lock::~Lock() {
8      m.unlock();
9  }

```

jun 27, 17 14:46

**ResultWindow.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_RESULTWINDOW_H
2  #define Z_TPGRUPAL_RESULTWINDOW_H
3
4
5  #include <gtkmm/window.h>
6  #include <gtkmm/builder.h>
7  #include <gtkmm/button.h>
8  #include <gtkmm/label.h>
9
10 class ResultWindow : public Gtk::Window {
11     Gtk::Button* menu;
12     Gtk::Button* close;
13     Gtk::Label* winner;
14     Gtk::Label* loser;
15
16     bool back_to_menu;
17 public:
18     ResultWindow(BaseObjectType *cobject,
19                 const Glib::RefPtr<Gtk::Builder> &builder);
20
21     void display_lose_screen();
22     void display_win_screen();
23
24     void menu_click();
25
26     void close_click();
27
28     void on_show();
29
30     bool go_back_to_menu();
31 };
32
33
34 #endif //Z_TPGRUPAL_RESULTWINDOW_H

```

jun 27, 17 14:46

## ResultWindow.cpp

Page 1/1

```

1  #include "ResultWindow.h"
2
3  ResultWindow::ResultWindow(BaseObjectType *cobject,
4                             const Glib::RefPtr<Gtk::Builder> &builder) :
5      Gtk::Window(cobject)
6  {
7
8      builder->get_widget("BackToMenuButton", menu);
9      builder->get_widget("CloseGameButton", close);
10     builder->get_widget("WinnerLabel", winner);
11     builder->get_widget("LoserLabel", loser);
12
13     menu->signal_clicked().connect(sigc::mem_fun(*this,
14                                                  &ResultWindow::menu_click));
15     close->signal_clicked().connect(sigc::mem_fun(*this,
16                                                  &ResultWindow::close_click));
17 }
18
19 void ResultWindow::display_lose_screen() {
20     winner->hide();
21 }
22
23 void ResultWindow::display_win_screen() {
24     loser->hide();
25 }
26
27 void ResultWindow::menu_click() {
28     back_to_menu = true;
29     this->hide();
30 }
31
32 void ResultWindow::close_click() {
33     back_to_menu = false;
34     this->hide();
35 }
36
37 bool ResultWindow::go_back_to_menu() {
38     return back_to_menu;
39 }
40
41 void ResultWindow::on_show() {
42     back_to_menu = false;
43     Gtk::Widget::on_show();
44 }
45
46

```

jun 27, 17 14:46

## MenuWindow.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_MENUWINDOW_H
2  #define Z_TPGRUPAL_MENUWINDOW_H
3
4
5  #include <gtkmm/window.h>
6  #include <gtkmm/builder.h>
7  #include <gtkmm/listbox.h>
8  #include <gtkmm/button.h>
9  #include <gtkmm/listviewtext.h>
10 #include <gtkmm/textview.h>
11 #include "../ServerMessenger.h"
12
13 class MenuWindow : public Gtk::Window {
14     Gtk::Label* available_lobbies;
15     Gtk::Button* join_button;
16     Gtk::Button* create_button;
17     Gtk::Entry* lobby_entry;
18     ServerMessenger* messenger;
19
20     bool joined_successfully;
21 public:
22     MenuWindow(BaseObjectType *cobject,
23               const Glib::RefPtr<Gtk::Builder> &builder);
24
25     void join_click();
26
27     void load_messenger(ServerMessenger* messenger);
28
29     void create_click();
30     void update_lobbies(const std::vector<std::string>& lobbies);
31
32     void on_show();
33
34     void join_lobby();
35     bool joined_lobby();
36 };
37
38
39 #endif //Z_TPGRUPAL_MENUWINDOW_H

```

jun 27, 17 14:46

## MenuWindow.cpp

Page 1/1

```

1 #include <gtkmm/label.h>
2 #include "MenuWindow.h"
3
4 MenuWindow::MenuWindow(BaseObjectType *cobject,
5                         const Glib::RefPtr<Gtk::Builder> &builder) :
6     Gtk::Window(cobject)
7 {
8     builder->get_widget("AvailableLobbies", available_lobbies);
9     builder->get_widget("CreateLobbyButton", create_button);
10    builder->get_widget("JoinLobbyButton", join_button);
11    builder->get_widget("LobbyEntry", lobby_entry);
12    join_button->signal_clicked().connect(
13        sigc::mem_fun(*this,
14                       &MenuWindow::join_click));
15
16    create_button->signal_clicked().connect(
17        sigc::mem_fun(*this,
18                       &MenuWindow::create_click));
19 }
20
21 void MenuWindow::join_click() {
22     std::string lobby = lobby_entry->get_text();
23     try {
24         std::stoi(lobby);
25     } catch(std::invalid_argument& e) {
26         std::cerr << "Invalid lobby ID. Insert only numbers!" << std::endl;
27         return;
28     }
29     messenger->send("joinlobby-" + lobby);
30 }
31
32 void MenuWindow::load_messenger(ServerMessenger *messenger) {
33     this->messenger = messenger;
34 }
35
36 void MenuWindow::create_click() {
37     messenger->send("createlobby");
38 }
39
40 void MenuWindow::update_lobbies(const std::vector<std::string> &lobbies) {
41     std::stringstream text;
42     for (const std::string& lobby : lobbies) {
43         text << lobby << std::endl;
44     }
45     available_lobbies->set_text(text.str());
46 }
47
48 bool MenuWindow::joined_lobby() {
49     return joined_successfully;
50 }
51
52 void MenuWindow::on_show() {
53     joined_successfully = false;
54     Gtk::Widget::on_show();
55 }
56
57 void MenuWindow::join_lobby() {
58     joined_successfully = true;
59     hide();
60 }
61

```

jun 27, 17 14:46

## LobbyWindow.h

Page 1/1

```

1 #ifndef Z_TPGRUPAL_LOBBYWINDOW_H
2 #define Z_TPGRUPAL_LOBBYWINDOW_H
3
4
5 #include <gtkmm/window.h>
6 #include <gtkmm/button.h>
7 #include <gtkmm/label.h>
8 #include <gtkmm/builder.h>
9 #include <gtkmm/togglebutton.h>
10 #include <gtkmm/entry.h>
11 #include <string>
12 #include <vector>
13
14 #define PLAYERS_AMT 4
15 #include "../ServerMessenger.h"
16
17 class LobbyWindow : public Gtk::Window {
18     Gtk::Button *start;
19     Gtk::Button *ready;
20     Gtk::Label *players[PLAYERS_AMT];
21     ServerMessenger *m;
22     Gtk::Label* maps_label;
23     Gtk::Entry* maps_entry;
24     std::string default_label;
25
26     bool started = false;
27
28 public:
29     LobbyWindow(BaseObjectType *cobject,
30                 const Glib::RefPtr<Gtk::Builder> &builder);
31
32     void set_messenger(ServerMessenger& m);
33     void update_player_name(int at, const std::string& name);
34
35     std::vector<std::string> get_player_names();
36     void start_game();
37     bool game_started();
38     void on_show();
39     void update_maps(const std::string& maps);
40
41 private:
42     void click_start();
43     void click_ready();
44 };
45
46
47 #endif //Z_TPGRUPAL_LOBBYWINDOW_H

```

jun 27, 17 14:46

## LobbyWindow.cpp

Page 1/2

```

1  #include <iostream>
2  #include "LobbyWindow.h"
3  #include "../ServerMessenger.h"
4
5  LobbyWindow::LobbyWindow(BaseObjectType *cobject,
6                          const Glib::RefPtr<Gtk::Builder> &builder) :
7      Gtk::Window(cobject)
8  {
9
10     builder->get_widget("PlayerStatus1", players[0]);
11     builder->get_widget("PlayerStatus2", players[1]);
12     builder->get_widget("PlayerStatus3", players[2]);
13     builder->get_widget("PlayerStatus4", players[3]);
14     builder->get_widget("StartGame", start);
15     builder->get_widget("ReadyButton", ready);
16     builder->get_widget("MapsLabel", maps_label);
17     builder->get_widget("MapsEntry", maps_entry);
18     default_label = players[0]->get_text();
19     start->signal_clicked().connect(sigc::mem_fun(*this,
20                                                  &LobbyWindow::click_start));
21
22     ready->signal_clicked().connect(sigc::mem_fun(*this,
23                                                  &LobbyWindow::click_ready));
24 }
25
26 void LobbyWindow::click_start() {
27     std::string map = maps_entry->get_text();
28     m->send("startgame-" + map);
29 }
30
31 void LobbyWindow::set_messenger(ServerMessenger &m) {
32     this->m = &m;
33 }
34
35 void LobbyWindow::update_player_name(int at, const std::string &name) {
36     if (at < 4) {
37         players[at]->set_text(name);
38     }
39 }
40
41 std::vector<std::string> LobbyWindow::get_player_names() {
42     std::vector<std::string> names;
43     for (Gtk::Label* player : players) {
44         std::string name = player->get_text();
45         if (name == default_label) {
46             name = "";
47         }
48         names.push_back(name);
49     }
50     return names;
51 }
52
53
54 void LobbyWindow::click_ready() {
55     if (ready->get_label() == "Ready") { // already pressed
56         m->send("ready");
57         ready->set_label("Unready");
58     } else if (ready->get_label() == "Unready") {
59         m->send("unready");
60         ready->set_label("Ready");
61     }
62 }
63
64 void LobbyWindow::start_game() {
65     started = true;
66     this->hide();

```

jun 27, 17 14:46

## LobbyWindow.cpp

Page 2/2

```

67 }
68
69 bool LobbyWindow::game_started() {
70     return started;
71 }
72
73 void LobbyWindow::update_maps(const std::string &maps) {
74     maps_label->set_text(maps);
75 }
76
77 void LobbyWindow::on_show() {
78     started = false;
79     Gtk::Widget::on_show();
80 }
81

```



jun 27, 17 14:46

## InitialWindow.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_INITIALWINDOW_H
2  #define Z_TPGRUPAL_INITIALWINDOW_H
3
4
5  #include <gtkmm/window.h>
6  #include <gtkmm/builder.h>
7  #include <gtkmm/entry.h>
8  #include <gtkmm/button.h>
9  #include <socket.h>
10 #include "../ServerMessenger.h"
11
12 class InitialWindow : public Gtk::Window {
13     Gtk::Entry *address_entry;
14     Gtk::Entry *port_entry;
15     Gtk::Entry *name_entry;
16     Gtk::Button *connect;
17     std::shared_ptr<ServerMessenger> messenger;
18     std::string name;
19
20 public:
21     InitialWindow(BaseObjectType *cobject,
22                 const Glib::RefPtr<Gtk::Builder> &builder);
23
24     std::shared_ptr<ServerMessenger> get_socket();
25
26     const std::string &get_username();
27
28 private:
29     void on_click();
30     void send_name();
31 };
32
33 #endif //Z_TPGRUPAL_INITIALWINDOW_H

```

jun 27, 17 14:46

## InitialWindow.cpp

Page 1/1

```

1  #include <iostream>
2  #include "InitialWindow.h"
3
4  #define ERROR_MSG "error"
5  #define OK_MSG "ok"
6
7  InitialWindow::InitialWindow(BaseObjectType *cobject,
8                             const Glib::RefPtr<Gtk::Builder> &builder) :
9      Gtk::Window(cobject) {
10     builder->get_widget("AddressEntry", address_entry);
11     builder->get_widget("PortEntry", port_entry);
12     builder->get_widget("NameEntry", name_entry);
13     builder->get_widget("ConnectButton", connect);
14     connect->signal_clicked().connect(sigc::mem_fun(*this,
15                                                    &InitialWindow::on_click));
16 }
17
18 void InitialWindow::on_click() {
19     std::string addr_str = address_entry->get_text();
20     std::string port_str = port_entry->get_text();
21     name = name_entry->get_text();
22     if (messenger.get()) { // Connection already established
23         send_name();
24         return;
25     }
26
27     try {
28         int port = 0;
29         port = std::stoi(port_str);
30         Socket s(addr_str.c_str(), port);
31         messenger = std::shared_ptr<ServerMessenger>(new ServerMessenger(s));
32         send_name();
33     } catch (SocketError &e) {
34         std::cerr << "Could not connect to specified addr/port" << std::endl;
35         return;
36     }
37 }
38
39 void InitialWindow::send_name() {
40     messenger.get()->send("changenname-" + name);
41     std::string response = messenger.get()->receive();
42     if (response == ERROR_MSG) {
43         std::cerr << "A player with this name already exists" << std::endl;
44         return;
45     }
46     hide();
47 }
48
49 std::shared_ptr<ServerMessenger> InitialWindow::get_socket() {
50     return messenger;
51 }
52
53 const std::string &InitialWindow::get_username() {
54     return name;
55 }
56
57

```

jun 27, 17 14:46	GameWindow.h	Page 1/2
1	<b>#ifndef</b> Z_TPGRUPAL_GAMEWINDOW_H	
2	<b>#define</b> Z_TPGRUPAL_GAMEWINDOW_H	
3		
4	<b>#include</b> <gtkmm/window.h>	
5	<b>#include</b> <gtkmm/grid.h>	
6	<b>#include</b> <gtkmm/button.h>	
7	<b>#include</b> <gtkmm/builder.h>	
8	<b>#include</b> <gtkmm/box.h>	
9	<b>#include</b> <gtkmm/label.h>	
10	<b>#include</b> <gtkmm/applicationwindow.h>	
11	<b>#include</b> "../GameArea.h"	
12	<b>#include</b> "../BuildingsMonitor.h"	
13	<b>#include</b> "../MapMonitor.h"	
14	<b>#include</b> "../ServerMessenger.h"	
15	<b>#include</b> "../panels/UnitPanel.h"	
16	<b>#include</b> "../panels/BuildingPanel.h"	
17		
18	class GameWindow : public Gtk::ApplicationWindow {	
19	GameArea *gameArea;	
20	Gtk::Box *panel;	
21	UnitPanel *unit_panel;	
22	BuildingPanel *building_panel;	
23	Gtk::Box *group_panel;	
24	Gtk::Label *panelLabel;	
25		
26	UnitsMonitor *unitsMonitor;	
27	BuildingsMonitor *buildingsMonitor;	
28	MapMonitor *mapMonitor;	
29	ServerMessenger *messenger;	
30		
31	Unit selected_unit;	
32	Building selected_building;	
33		
34	bool unit_selection;	
35	bool building_selection;	
36		
37	std::string me;	
38	public:	
39	GameWindow(BaseObjectType *cobject,	
40	<b>const</b> Glib::RefPtr<Gtk::Builder> &builder);	
41		
42	virtual ~GameWindow();	
43		
44	/**	
45	* Saves resources' monitors and passes them to the gameArea.	
46	*/	
47	void	
48	setResources(UnitsMonitor *unitsMonitor, BuildingsMonitor *buildingsMonitor,	
49	MapMonitor *mapMonitor, ServerMessenger *messenger,	
50	<b>const</b> std::string &owner);	
51		
52	void setMapData();	
53	void factory_change_unit(std::string &path);	
54		
55	void update_factory_panel( <b>const</b> std::string& type, int fire_rate, int hp);	
56	void update_factory_timer(int minutes, int seconds);	
57		
58	void update_name( <b>const</b> std::string& name);	
59		
60	protected:	
61	bool onTimeout();	
62		
63	private:	
64	void update_side_panels();	
65	// Functions to change the window's side panel	
66	bool change_view_to_unit();	

jun 27, 17 14:46	GameWindow.h	Page 2/2
67		
68	bool change_view_to_building();	
69		
70	bool change_view_to_unit_group();	
71		
72	bool on_button_release_event(GdkEventButton *event);	
73		
74	void factory_next();	
75		
76	void factory_create_unit();	
77		
78	void process_attack();	
79		
80	bool on_key_press_event(GdkEventKey *event) override;	
81		
82	void remove_side_panel();	
83		
84	void factory_prev();	
85		
86	void process_movement() <b>const</b> ;	
87	};	
88		
89	<b>#endif</b> //Z_TPGRUPAL_GAMEWINDOW_H	

jun 27, 17 14:46

## GameWindow.cpp

Page 1/4

```

1  #include <iostream>
2  #include <giomm.h>
3  #include <gdkmm.h>
4  #include "GameWindow.h"
5
6  #define SCREENWIDTH 1200
7  #define SCREENHEIGHT 800
8  #define FRAMERATE 10    //fps
9
10 GameWindow::GameWindow(BaseObjectType *cobject,
11                          const Glib::RefPtr<Gtk::Builder> &builder) :
12     Gtk::ApplicationWindow(cobject)
13 {
14     builder->get_widget_derived("GameArea", gameArea);
15
16     gameArea->set_size_request(SCREENWIDTH * 6 / 7, SCREENHEIGHT);
17     builder->get_widget("SidePanel", panel);
18     builder->get_widget_derived("BuildingView", building_panel);
19     builder->get_widget_derived("UnitView", unit_panel);
20     builder->get_widget("GroupView", group_panel);
21     builder->get_widget("PanelDisplayLabel", panelLabel);
22
23
24     building_panel->next_button()->
25         signal_clicked().connect(
26             sigc::mem_fun(*this, &GameWindow::factory_next));
27
28     building_panel->create_button()->
29         signal_clicked().connect(
30             sigc::mem_fun(*this, &GameWindow::factory_create_unit));
31
32
33     building_panel->prev_button()->
34         signal_clicked().connect(
35             sigc::mem_fun(*this, &GameWindow::factory_prev));
36     // Logic for redrawing the map every frame
37     sigc::slot<bool> mySlot = sigc::mem_fun(*this, &GameWindow::onTimeout);
38     Glib::signal_timeout().connect(mySlot, 1000 / FRAMERATE);
39
40     show_all_children();
41     add_events(Gdk::EventMask::KEY_PRESS_MASK);
42 }
43
44 GameWindow::~GameWindow() {
45 }
46
47 bool GameWindow::onTimeout() {
48     // force our program to redraw the entire thing
49     auto win = get_window();
50     if (win) {
51         Gdk::Rectangle r(0, 0, get_allocation().get_width(),
52                          get_allocation().get_height());
53         win->invalidate_rect(r, false);
54     }
55     update_side_panels();
56     return true;
57 }
58
59 void GameWindow::setResources(UnitsMonitor *unitsMonitor,
60                               BuildingsMonitor *buildingsMonitor,
61                               MapMonitor *mapMonitor,
62                               ServerMessenger *messenger,
63                               const std::string &owner) {
64     this->unitsMonitor = unitsMonitor;
65     this->buildingsMonitor = buildingsMonitor;
66     this->mapMonitor = mapMonitor;

```

jun 27, 17 14:46

## GameWindow.cpp

Page 2/4

```

67     this->messenger = messenger;
68     gameArea->setResources(unitsMonitor, buildingsMonitor, mapMonitor, owner);
69 }
70
71
72 bool GameWindow::change_view_to_unit() {
73     for (auto child : panel->get_children()) {
74         child->hide();
75     }
76
77     unit_panel->show();
78     panelLabel->set_text(unit_panel->get_label());
79     unit_panel->update_portrait(selected_unit.getType(),
80                               selected_unit.getTeam());
81     return true;
82 }
83
84 bool GameWindow::change_view_to_building() {
85     for (auto child : panel->get_children()) {
86         child->hide();
87     }
88     building_panel->show();
89     panelLabel->set_text(building_panel->get_label());
90
91     return true;
92 }
93
94 bool GameWindow::change_view_to_unit_group() {
95     for (auto child : panel->get_children()) {
96         child->hide();
97     }
98
99     group_panel->show();
100
101     return true;
102 }
103
104
105 bool GameWindow::on_button_release_event(GdkEventButton *event) {
106     if (event->button == GDK_BUTTON_SECONDARY) {
107         if (!(selected_unit.get_owner() == me)) {
108             return true;
109         }
110
111         if (gameArea->unit_selected() || gameArea->buildings_selected()) {
112             // We already are selecting an unit, process attack
113             process_attack();
114         } else { // Click on empty place, movement
115             process_movement();
116         }
117     } else if (event->button == GDK_BUTTON_PRIMARY) {
118         if (gameArea->buildings_selected()) { // New building selected
119             selected_building = buildingsMonitor->get_selected().at(0);
120             messenger->send(
121                 "factory-" + std::to_string(selected_building.get_ID())
122                 + "-current");
123
124             change_view_to_building();
125
126             // Change selection status
127             unit_selection = false;
128             building_selection = true;
129         } else if (gameArea->unit_selected()) { // New unit selected
130             selected_unit = unitsMonitor->getSelectedUnits().at(0);
131             change_view_to_unit();
132 }

```

jun 27, 17 14:46

GameWindow.cpp

Page 3/4

```

133         // Change selection status
134         building_selection = false;
135         unit_selection = true;
136     }
137 }
138 return true;
139 }
140
141 void GameWindow::process_movement() const {
142     int id = selected_unit.get_ID();
143     std::pair<int, int> coords = gameArea->get_coords();
144     int x = coords.first;
145     int y = coords.second;
146
147     std::stringstream s;
148     int flag = mapMonitor->get_flag_at(x, y);
149     if (flag > 0) { // Issue a flag grabbing cmd to move towards the position
150         s << "grab-" << id << "-" << flag;
151     } else {
152         s << "mv-" << id << "-" << x << "-" << y;
153     };
154     messenger->send(s.str());
155 }
156
157 void GameWindow::factory_next() {
158     int id = selected_building.get_ID();
159     messenger->send("factory-"+std::to_string(id)+"-next");
160 }
161
162 void GameWindow::factory_change_unit(std::string &path) {
163     building_panel->change_unit(path);
164 }
165
166 void GameWindow::factory_create_unit() {
167     int id = selected_building.get_ID();
168     messenger->send("factory-"+std::to_string(id)+"-create");
169 }
170
171 void GameWindow::setMapData() {
172     gameArea->setMapData();
173 }
174
175 void GameWindow::process_attack() {
176     std::vector<Unit> units = unitsMonitor->getSelectedUnits();
177     std::string target;
178     if (units.size()) { // other unit selected
179         Unit other = units.at(0);
180         if (selected_unit.getTeam() == other.getTeam()) {
181             return;
182         }
183         target = std::to_string(other.get_ID());
184     }
185     std::vector<Building> buildings = buildingsMonitor->get_selected();
186     if (buildings.size()) {
187         Building other = buildings.at(0);
188         if (selected_unit.getTeam() == other.getTeam()) {
189             return;
190         }
191         target = std::to_string(other.get_ID());
192     }
193     std::string attack = "atk-" + std::to_string(selected_unit.get_ID()) +
194         "-" + target;
195     messenger->send(attack);
196 }

```

jun 27, 17 14:46

GameWindow.cpp

Page 4/4

```

199 }
200
201 void GameWindow::update_name(const std::string &name) {
202     me = name;
203 }
204
205 bool GameWindow::on_key_press_event(GdkEventKey *event) {
206     // Clear selection
207     if (event->keyval == GDK_KEY_Escape) {
208         remove_side_panel();
209     }
210     return Gtk::Window::on_key_press_event(event);
211 }
212
213 void GameWindow::remove_side_panel() {
214     selected_building = Building();
215     selected_unit = Unit();
216     for (auto child : panel->get_children()) {
217         child->hide();
218     }
219     panelLabel->set_text("Z");
220 }
221
222 void GameWindow::update_side_panels() {
223     if (!unit_selection ^ !building_selection) {
224         remove_side_panel();
225     } else if (unit_selection) {
226         int unit_id = selected_unit.get_ID();
227         // UpdateUnit the unit reference
228         selected_unit = unitsMonitor->get_unit(unit_id);
229         unit_panel->set_name(selected_unit.get_unit_name());
230         unit_panel->set_owner(selected_unit.get_owner());
231         unit_panel->set_max_hp(selected_unit.get_max_hp());
232         unit_panel->set_hp(selected_unit.get_hp());
233     } else if (building_selection) {
234         int building_id = selected_building.get_ID();
235
236         selected_building = buildingsMonitor->get_building(building_id);
237         building_panel->set_max_hp(selected_building.get_max_hp());
238         building_panel->set_hp(selected_building.get_hp());
239         building_panel->set_owner(selected_building.get_owner());
240         building_panel->set_time_left(selected_building.get_time_left());
241     }
242 }
243
244 void GameWindow::factory_prev() {
245     int id = selected_building.get_ID();
246     messenger->send("factory-"+std::to_string(id)+"-prev");
247 }
248
249 void GameWindow::update_factory_panel(const std::string &type, int fire_rate,
250     int hp) {
251     building_panel->set_unit_hp(hp);
252     building_panel->set_unit_fire_rate(fire_rate);
253     building_panel->set_unit_type(type, selected_building.getTeam());
254 }
255
256 void GameWindow::update_factory_timer(int minutes, int seconds) {
257     building_panel->set_time_left(std::pair<int, int>(minutes, seconds));
258 }

```

jun 27, 17 14:46

## UnitsMonitor.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_UNITSMONITOR_H
2  #define Z_TPGRUPAL_UNITSMONITOR_H
3
4
5  #include <mutex>
6  #include "Unit.h"
7
8  class UnitsMonitor {
9      std::vector<Unit> units;
10     std::mutex m;
11
12 public:
13
14     void addUnit(Unit &unit);
15
16     void removeUnit(int id);
17
18     void update_position(int unit_id, int x, int y);
19
20     std::vector<Unit>
21     getUnitsToDraw(unsigned int minX, unsigned int maxX, unsigned int minY,
22                   unsigned int maxY);
23
24     void
25     markAsSelectedInRange(bool &unitsSelected, gdouble xStartCoordinate,
26                          gdouble yStartCoordinate, gdouble xFinishCoordinate,
27                          gdouble yFinishCoordinate);
28
29     void wipeSelected();
30
31     std::vector<Unit> getSelectedUnits();
32
33     void update_position(int id, ActionsEnum state, int x, int y);
34     void update_health(int id, unsigned int hp);
35
36     Unit get_unit(int id);
37
38     void clear();
39 };
40
41
42 #endif //Z_TPGRUPAL_UNITSMONITOR_H

```

jun 27, 17 14:46

## UnitsMonitor.cpp

Page 1/2

```

1  #include <Lock.h>
2  #include <iostream>
3  #include "UnitsMonitor.h"
4  #include <vector>
5
6
7  void UnitsMonitor::addUnit(Unit &unit) {
8      Lock l(m);
9      units.push_back(unit);
10 }
11
12 void UnitsMonitor::removeUnit(int id) {
13     Lock l(m);
14     for (auto unit = units.begin(); unit != units.end(); ++unit) {
15         if (unit->get_ID() == id) {
16             units.erase(unit);
17         }
18     }
19 }
20
21 void UnitsMonitor::update_position(int unit_id, int x, int y) {
22     Lock l(m);
23     for (auto unit = units.begin(); unit != units.end(); ++unit) {
24         if (unit->get_ID() == unit_id) {
25             unit->update_position(x, y);
26         }
27     }
28 }
29
30 void UnitsMonitor::update_position(int id, ActionsEnum state, int x, int y) {
31     Lock l(m);
32     for (auto unit = units.begin(); unit != units.end(); ++unit) {
33         if (unit->get_ID() == id) {
34             unit->update_position(x, y);
35             unit->update_state(state);
36         }
37     }
38 }
39
40 std::vector<Unit>
41 UnitsMonitor::getUnitsToDraw(unsigned int minX, unsigned int maxX,
42                             unsigned int minY, unsigned int maxY) {
43     Lock l(m);
44     std::vector<Unit> returnVector;
45
46     for (Unit &unit : units) {
47         if (unit.getXCoordinate() >= minX and
48             unit.getXCoordinate() <= maxX and
49             unit.getYCoordinate() >= minY and
50             unit.getYCoordinate() <= maxY) {
51             returnVector.emplace_back(unit);
52         }
53     }
54     return returnVector;
55 }
56
57 void
58 UnitsMonitor::markAsSelectedInRange(bool &unitsSelected,
59                                     gdouble xStartCoordinate,
60                                     gdouble yStartCoordinate,
61                                     gdouble xFinishCoordinate,
62                                     gdouble yFinishCoordinate) {
63     Lock l(m);
64     for (Unit &unit : units) {
65         unit.markAsSelectedInRange(unitsSelected, xStartCoordinate,
66                                   yStartCoordinate, xFinishCoordinate,

```

jun 27, 17 14:46

## UnitsMonitor.cpp

Page 2/2

```

67         yFinishCoordinate);
68         /* if unit selected, break because we only want to select one unit */
69         if (unitsSelected)
70             break;
71     }
72 }
73
74 std::vector<Unit> UnitsMonitor::getSelectedUnits() {
75     std::vector<Unit> units;
76     for (Unit &unit : this->units) {
77         if (unit.is_selected()) {
78             units.push_back(unit);
79         }
80     }
81     return units;
82 }
83
84 void UnitsMonitor::wipeSelected() {
85     for (Unit &unit : units) {
86         unit.unselect();
87     }
88 }
89
90 void UnitsMonitor::update_health(int id, unsigned int hp) {
91     Lock l(m);
92     for (auto unit = units.begin(); unit != units.end(); ++unit) {
93         if (unit->get_ID() == id) {
94             if (!hp) {
95                 units.erase(unit);
96                 break;
97             } else {
98                 unit->update_hp(hp);
99             }
100         }
101     }
102 }
103
104 Unit UnitsMonitor::get_unit(int id) {
105     for (Unit& unit: units) {
106         if (unit.get_ID() == id){
107             return unit;
108         }
109     }
110     return Unit();
111 }
112
113 void UnitsMonitor::clear() {
114     Lock l(m);
115     units.clear();
116 }

```

jun 27, 17 14:46

## Unit.h

Page 1/2

```

1  #ifndef Z_TPGRUPAL_UNIT_H
2  #define Z_TPGRUPAL_UNIT_H
3
4
5  #include <vector>
6  #include <string>
7  #include <map>
8  #include "gtkmm/drawingarea.h"
9  #include "Armament.h"
10 #include "enums/TeamEnum.h"
11 #include "enums/ActionsEnum.h"
12 #include "enums/UnitsEnum.h"
13 #include "enums/RotationsEnum.h"
14 #include <utility>
15 #include <mutex>
16
17 class Unit {
18 private:
19     int id;
20
21     /* unitType can be: robot, vehicle, tank */
22     UnitsEnum unitType;
23
24     std::string unit_name;
25
26     ActionsEnum actionType;
27
28     RotationsEnum rotation;
29
30     Armament armament;
31
32     /* freq. with which the unit shoots */
33     unsigned short frequency;
34
35     unsigned int lifeLeft;
36
37     unsigned int totalLife;
38
39     unsigned short velocity;
40
41     std::pair<unsigned int, unsigned int> position;
42     std::pair<unsigned int, unsigned int> prev_position;
43
44     /* bool selected: indicates weather the unit has been selected
45      * with the mouse or not */
46     bool selected;
47
48     /* bool that indicates whether the unit is shooting or not */
49     bool shooting;
50
51     /* indicates to which getTeam the unit belongs */
52     TeamEnum team;
53
54     /* counters to know which img of the ones that conform an animation
55      * should be drawn*/
56     unsigned short shootingDrawingCounter;
57     unsigned short standingDrawingCounter;
58     unsigned short walkingDrawingCounter;
59
60     std::string owner;
61 public:
62     Unit(int id, std::pair<unsigned int, unsigned int> position,
63         UnitsEnum unitType, TeamEnum team, unsigned int hp);
64
65     Unit();
66     int get_ID() const;

```

jun 27, 17 14:46

## Unit.h

Page 2/2

```

67     void update_position(int x, int y);
68
69     void update_state(ActionsEnum state);
70
71     void markAsSelectedInRange(bool &unitsSelected, gdouble xStartCoordinate,
72                               gdouble yStartCoordinate,
73                               gdouble xFinishCoordinate,
74                               gdouble yFinishCoordinate);
75
76     bool isShooting();
77
78     TeamEnum getTeam();
79
80     RotationsEnum getRotation();
81
82     UnitsEnum getType();
83
84     ActionsEnum getAction();
85
86     unsigned int getXCoordinate();
87
88     unsigned int getYCoordinate();
89
90     bool is_selected();
91
92     void unselect();
93
94     void update_rotation();
95
96     void update_owner(const std::string& owner);
97
98     void update_unit_name(const std::string& name);
99
100    void update_hp(unsigned int hp);
101    std::string get_owner();
102    int get_max_hp();
103    int get_hp();
104    std::string get_unit_name();
105 };
106
107
108
109 #endif //Z_TPGRUPAL_UNIT_H

```

jun 27, 17 14:46

## Unit.cpp

Page 1/3

```

1  #include "Unit.h"
2  #include <iostream>
3
4  #define IMG_SIZE_IN_PX 4
5  #define TEAM_NEUTRAL "None"
6  Unit::Unit(int id, std::pair<unsigned int, unsigned int> position,
7            UnitsEnum unitType, TeamEnum team, unsigned int hp)
8      : id(id), position(position), unitType(unitType),
9        rotation(RotationsEnum::r090), team(team),
10        selected(false), actionType(ActionsEnum::STAND),
11        owner(TEAM_NEUTRAL), totalLife(hp), lifeLeft(hp)
12 {
13     /* units initial rotation is facing 'to the player'; action: standing. */
14     //todo check what else should be initialized
15 }
16
17 int Unit::get_ID() const {
18     return id;
19 }
20
21 void Unit::update_position(int x, int y) {
22     std::pair<int, int> new_pos(x, y);
23
24     prev_position = position;
25     position = new_pos;
26     update_rotation();
27 }
28
29 void Unit::update_rotation() {
30     auto prev_x = prev_position.first;
31     auto prev_y = prev_position.second;
32     auto x = position.first;
33     auto y = position.second;
34
35     if (x == prev_x and y < prev_y) {
36         rotation = RotationsEnum::r090;
37     } else if (x > prev_x and y < prev_y) {
38         rotation = RotationsEnum::r045;
39     } else if (x > prev_x and y == prev_y) {
40         rotation = RotationsEnum::r000;
41     } else if (x > prev_x and y > prev_y) {
42         rotation = RotationsEnum::r315;
43     } else if (x == prev_x and y > prev_y) {
44         rotation = RotationsEnum::r270;
45     } else if (x < prev_x and y > prev_y) {
46         rotation = RotationsEnum::r225;
47     } else if (x < prev_x, y == prev_y) {
48         rotation = RotationsEnum::r180;
49     } else if (x < prev_x and y < prev_y) {
50         rotation = RotationsEnum::r135;
51     }
52 }
53
54 void Unit::update_state(ActionsEnum state) {
55     this->actionType = state;
56 }
57
58 void
59 Unit::markAsSelectedInRange(bool &unitsSelected, gdouble xStartCoordinate,
60                             gdouble yStartCoordinate,
61                             gdouble xFinishCoordinate,
62                             gdouble yFinishCoordinate) {
63     if (position.first >= xStartCoordinate - IMG_SIZE_IN_PX ^
64         position.first <= xFinishCoordinate + IMG_SIZE_IN_PX ^
65         position.second >= yStartCoordinate - IMG_SIZE_IN_PX ^
66         position.second <= yFinishCoordinate + IMG_SIZE_IN_PX) {

```

jun 27, 17 14:46

Unit.cpp

Page 2/3

```

67         selected = true;
68         unitsSelected = true;
69     }
70 }
71
72 bool Unit::isShooting() {
73     return shooting;
74 }
75
76 TeamEnum Unit::getTeam() {
77     return team;
78 }
79
80 RotationsEnum Unit::getRotation() {
81     return rotation;
82 }
83
84 unsigned int Unit::getXCoordinate() {
85     return position.first;
86 }
87
88 unsigned int Unit::getYCoordinate() {
89     return position.second;
90 }
91
92 UnitsEnum Unit::getType() {
93     return unitType;
94 }
95
96 ActionsEnum Unit::getAction() {
97     return actionType;
98 }
99
100 bool Unit::is_selected() {
101     return selected;
102 }
103
104 void Unit::unselect() {
105     selected = false;
106 }
107
108 Unit::Unit() {
109     id = 0;
110 }
111
112 void Unit::update_owner(const std::string &owner) {
113     this->owner = owner;
114 }
115
116 std::string Unit::get_owner() {
117     return owner;
118 }
119
120 int Unit::get_max_hp() {
121     return totalLife;
122 }
123
124 int Unit::get_hp() {
125     return lifeLeft;
126 }
127
128 void Unit::update_unit_name(const std::string &name) {
129     unit_name = name;
130 }
131
132 std::string Unit::get_unit_name() {

```

jun 27, 17 14:46

Unit.cpp

Page 3/3

```

133     return unit_name;
134 }
135
136 void Unit::update_hp(unsigned int hp) {
137     lifeLeft = hp;
138 }

```



jun 27, 17 14:46

## ServerMessenger.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_SERVERMESSENGER_H
2  #define Z_TPGRUPAL_SERVERMESSENGER_H
3
4  #include <mutex>
5  #include <string>
6  #include <messenger.h>
7  #include "../common/socket.h"
8
9  /* Handles communication with the server */
10 class ServerMessenger {
11     Messenger messenger;
12     std::mutex send_m, recv_m;
13 public:
14     /* Constructor: connects to the given addr/port combination */
15     explicit ServerMessenger(Socket &s);
16
17     void send(const std::string &message);
18
19     std::string receive();
20
21     void kill();
22
23     ServerMessenger(ServerMessenger& other);
24 };
25
26 #endif //Z_TPGRUPAL_SERVERMESSENGER_H

```

jun 27, 17 14:46

## ServerMessenger.cpp

Page 1/1

```

1
2  #include <string>
3  #include "ServerMessenger.h"
4  #include "../common/Lock.h"
5
6  ServerMessenger::ServerMessenger(Socket &s) :
7      messenger(s) {
8  }
9
10 void ServerMessenger::send(const std::string &message) {
11     Lock l(send_m);
12     messenger.sendMessage(message);
13 }
14
15 std::string ServerMessenger::receive() {
16     Lock l(recv_m);
17     return messenger.recieveMessage();
18 }
19
20 void ServerMessenger::kill() {
21     messenger.shutdown();
22 }
23
24 ServerMessenger::ServerMessenger(ServerMessenger &other) :
25     messenger(other.messenger)
26 {
27 }

```

jun 27, 17 14:46

## UnitPanel.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_UNITPANEL_H
2  #define Z_TPGRUPAL_UNITPANEL_H
3
4
5  #include <gtkmm/box.h>
6  #include <gtkmm/builder.h>
7  #include <gtkmm/button.h>
8  #include <gtkmm/label.h>
9  #include <gtkmm/image.h>
10 #include "../enums/UnitsEnum.h"
11 #include "../enums/TeamEnum.h"
12
13 class UnitPanel : public Gtk::Box {
14     Gtk::Label *owner;
15     Gtk::Label *max_hp_label;
16     Gtk::Label *hp_label;
17     Gtk::Label *name_label;
18     Gtk::Image* portrait;
19 public:
20     UnitPanel(BaseObjectType *cobject,
21               const Glib::RefPtr<Gtk::Builder> &builder);
22
23     std::string get_label();
24
25     void set_name(std::string name);
26
27     void set_hp(int hp);
28
29     void set_max_hp(int hp);
30
31     void set_owner(const std::string& owner);
32
33     void update_portrait(UnitsEnum unit, TeamEnum team);
34 };
35
36
37 #endif //Z_TPGRUPAL_UNITPANEL_H

```

jun 27, 17 14:46

## UnitPanel.cpp

Page 1/2

```

1  #include <iostream>
2  #include "UnitPanel.h"
3  #include "../enums/UnitsEnum.h"
4  #include "../enums/TeamEnum.h"
5
6  #define PORTRAITS "res/portraits/"
7  const std::map<UnitsEnum, std::string> units = {
8      {UnitsEnum::GRUNT, std::string("grunt")},
9      {UnitsEnum::PSYCHO, std::string("psycho")},
10     {UnitsEnum::TOUGH, std::string("tough")},
11     {UnitsEnum::PYRO, std::string("pyro")},
12     {UnitsEnum::SNIPER, std::string("sniper")},
13     {UnitsEnum::LASER, std::string("laser")},
14     {UnitsEnum::GENERIC_ROBOT, std::string("generic_robot")},
15     {UnitsEnum::JEEP, std::string("jeep")},
16     {UnitsEnum::MEDIUM_TANK, std::string("medium_tank")},
17     {UnitsEnum::LIGHT_TANK, std::string("light_tank")},
18     {UnitsEnum::HEAVY_TANK, std::string("heavy_tank")},
19     {UnitsEnum::MML, std::string("mml")}
20 };
21
22 const std::map<TeamEnum, std::string> teams = {
23     {TeamEnum::BLUE, "blue"},
24     {TeamEnum::GREEN, "green"},
25     {TeamEnum::RED, "red"},
26     {TeamEnum::YELLOW, "yellow"}
27 };
28
29 UnitPanel::UnitPanel(BaseObjectType *cobject,
30                       const Glib::RefPtr<Gtk::Builder> &builder) :
31     Gtk::Box(cobject)
32 {
33     builder->get_widget("OwnerLabel", owner);
34     builder->get_widget("MaxHPLabel", max_hp_label);
35     builder->get_widget("HPLabel", hp_label);
36     builder->get_widget("NameLabel", name_label);
37     builder->get_widget("Portrait", portrait);
38
39 }
40
41 std::string UnitPanel::get_label() {
42     return "Unit";
43 }
44
45 void UnitPanel::set_name(std::string name) {
46     name_label->set_text(name);
47 }
48
49 void UnitPanel::set_hp(int hp) {
50     hp_label->set_text(std::to_string(hp));
51 }
52
53 void UnitPanel::set_max_hp(int hp) {
54     max_hp_label->set_text(std::to_string(hp));
55 }
56
57 void UnitPanel::set_owner(const std::string &owner) {
58     this->owner->set_text(owner);
59 }
60
61 void UnitPanel::update_portrait(UnitsEnum unit, TeamEnum team) {
62     std::string unit_name = units.find(unit)->second;
63     std::string color = teams.find(team)->second;
64     std::string path = PORTRAITS + unit_name + "_" + color + ".png";
65     portrait->set(path);
66     set_name(unit_name);

```

jun 27, 17 14:46

## UnitPanel.cpp

Page 2/2

```

67 }
68

```

jun 27, 17 14:46

## BuildingPanel.h

Page 1/2

```

1  #ifndef Z_TPGRUPAL_BUILDINGPANEL_H
2  #define Z_TPGRUPAL_BUILDINGPANEL_H
3
4
5  #include <gtkmm/box.h>
6  #include <gtkmm/image.h>
7  #include <gtkmm/builder.h>
8  #include <string>
9  #include <gtkmm/button.h>
10 #include "../enums/TeamEnum.h"
11
12 class BuildingPanel : public Gtk::Box {
13     Gtk::Image* building;
14     Gtk::Image* unit;
15     const std::string label = "Factory";
16     Gtk::Button* prev;
17     Gtk::Button* next;
18     Gtk::Button* create;
19
20     Gtk::Label* max_hp_label;
21     Gtk::Label* hp_label;
22     Gtk::Label* owner_label;
23     Gtk::Label* unit_type;
24     Gtk::Label* unit_hp;
25     Gtk::Label* unit_fire_rate;
26
27     Gtk::Label* minutes;
28     Gtk::Label* seconds;
29
30     int max_hp;
31     int hp;
32     std::string owner;
33     std::string type;
34     int fire_rate;
35     std::pair<int, int> time_left;
36 public:
37     BuildingPanel(BaseObjectType* cobject,
38                 const Glib::RefPtr<Gtk::Builder>& builder);
39
40     const std::string& get_label();
41     Gtk::Button* next_button();
42     Gtk::Button* create_button();
43
44     void change_unit(std::string& path);
45
46
47     void set_hp(int hp);
48
49     void set_max_hp(int hp);
50
51     void set_owner(const std::string& owner);
52
53     void set_unit_hp(int hp);
54
55     void set_unit_type(const std::string& type, TeamEnum team);
56
57     void set_unit_fire_rate(int fire_rate);
58
59     void set_time_left(std::pair<int, int> time);
60     Gtk::Button * prev_button();
61
62 protected:
63     bool update_labels();
64 };
65
66

```

jun 27, 17 14:46

**BuildingPanel.h**

Page 2/2

```

67 #endif //Z_TPGRUPAL_BUILDINGPANEL_H

```

jun 27, 17 14:46

**BuildingPanel.cpp**

Page 1/2

```

1  #include <giomm.h>
2  #include "BuildingPanel.h"
3  #include "../windows/GameWindow.h"
4  #define PORTRAITS_PATH "res/portraits/"
5
6  #define TIMEOUT 100
7
8  BuildingPanel::BuildingPanel(BaseObjectType *cobject,
9                               const Glib::RefPtr<Gtk::Builder> &builder) :
10      Gtk::Box(cobject) {
11      builder->get_widget("FactoryImage", building);
12      builder->get_widget("FactoryUnitImage", unit);
13      builder->get_widget("PrevUnitButton", prev);
14      builder->get_widget("NextUnitButton", next);
15      builder->get_widget("FactoryCreateButton", create);
16
17      builder->get_widget("FactoryMaxHPLabel", max_hp_label);
18      builder->get_widget("FactoryHPLabel", hp_label);
19      builder->get_widget("FactoryOwnerLabel", owner_label);
20
21      builder->get_widget("FactoryUnitHPLabel", unit_hp);
22      builder->get_widget("FactoryFireRateLabel", unit_fire_rate);
23      builder->get_widget("FactoryTypeLabel", unit_type);
24
25      builder->get_widget("FactoryMinutesLabel", minutes);
26      builder->get_widget("FactorySecondsLabel", seconds);
27      building->set("res/buildings/base_city.png");
28      Glib::signal_timeout().connect(sigc::mem_fun(*this,
29                                                    &BuildingPanel::update_labels),
30                                     TIMEOUT);
31  }
32
33
34  const std::map<TeamEnum, std::string> teams = {
35      {TeamEnum::NEUTRAL, "blue"},
36      {TeamEnum::BLUE, "blue"},
37      {TeamEnum::GREEN, "green"},
38      {TeamEnum::RED, "red"},
39      {TeamEnum::YELLOW, "yellow"}
40  };
41
42  const std::string &BuildingPanel::get_label() {
43      return label;
44  }
45
46  Gtk::Button *BuildingPanel::next_button() {
47      return next;
48  }
49
50  void BuildingPanel::change_unit(std::string &path) {
51      unit->set(path);
52  }
53
54  Gtk::Button *BuildingPanel::create_button() {
55      return create;
56  }
57
58  void BuildingPanel::set_hp(int hp) {
59      this->hp = hp;
60  }
61
62  void BuildingPanel::set_max_hp(int hp) {
63      this->max_hp = hp;
64  }
65
66  void BuildingPanel::set_owner(const std::string &owner) {

```

jun 27, 17 14:46

**BuildingPanel.cpp**

Page 2/2

```

67     this→owner = owner;
68 }
69
70 void BuildingPanel::set_unit_hp(int hp) {
71     unit_hp→set_text(std::to_string(hp));
72 }
73
74 void BuildingPanel::set_unit_type(const std::string &type, TeamEnum team) {
75     std::string color = teams.find(team)→second;
76     this→type = type;
77     unit→set(PORTRAITS_PATH + type + "_" + color + ".png");
78 }
79
80 void BuildingPanel::set_unit_fire_rate(int fire_rate) {
81     this→fire_rate = fire_rate;
82 }
83
84 Gtk::Button* BuildingPanel::prev_button() {
85     return prev;
86 }
87
88 void BuildingPanel::set_time_left(std::pair<int, int> time) {
89     this→time_left = time;
90 }
91
92 bool BuildingPanel::update_labels() {
93     hp_label→set_text(std::to_string(hp));
94     max_hp_label→set_text(std::to_string(max_hp));
95     minutes→set_text(std::to_string(time_left.first));
96     seconds→set_text(std::to_string(time_left.second));
97     unit_fire_rate→set_text(std::to_string(fire_rate));
98     unit_type→set_text(type);
99     return true;
100 }

```

jun 27, 17 14:46

**Nature.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_NATURE_H
2  #define Z_TPGRUPAL_NATURE_H
3
4
5  #include <utility>
6  #include "enums/NatureEnum.h"
7
8  class Nature {
9      NatureEnum type;
10     std::pair<unsigned int, unsigned int> position;
11     int id;
12
13 public:
14     Nature(NatureEnum type, std::pair<unsigned int, unsigned int> position,
15           int id);
16
17     std::pair<unsigned int, unsigned int> getPosition();
18
19     NatureEnum getType();
20 };
21
22
23 #endif //Z_TPGRUPAL_NATURE_H

```

jun 27, 17 14:46

**Nature.cpp**

Page 1/1

```

1  #include "Nature.h"
2
3  Nature::Nature(NatureEnum type, std::pair<unsigned int, unsigned int> position,
4                int id) :
5      type(type), position(position), id(id){
6  }
7
8  std::pair<unsigned int, unsigned int> Nature::getPosition() {
9      return position;
10 }
11
12 NatureEnum Nature::getType() {
13     return type;
14 }

```

jun 27, 17 14:46

**MapMonitor.h**

Page 1/2

```

1  #ifndef Z_TPGRUPAL_MAPMONITOR_H
2  #define Z_TPGRUPAL_MAPMONITOR_H
3
4
5  #include <mutex>
6  #include "Map.h"
7  #include "enums/TeamEnum.h"
8  #include <string>
9
10 class MapMonitor {
11 private:
12     Map map;
13     std::vector<std::string> players;
14
15     std::mutex m;
16
17     bool winner = false;
18     bool loser = false;
19 public:
20
21     void setCell(unsigned int xCoordinate,
22                 unsigned int yCoordinate,
23                 std::string terrainType);
24
25     void initializeMap(unsigned int xSize, unsigned int ySize);
26
27     unsigned int getXSize();
28
29     unsigned int getYSize();
30
31     std::string getTerrainTypeAt(unsigned int xCoordinate,
32                                 unsigned int yCoordinate);
33
34     void addNature(Nature nature);
35
36     void markAsSelectedInRange(bool& terrainSelected, double xStartCoordinate,
37                               double yStartCoordinate,
38                               double xFinishCoordinate,
39                               double yFinishCoordinate);
40
41     std::vector<Nature>
42     getNatureToDraw(unsigned int minX, unsigned int maxX, unsigned int minY,
43                     unsigned int maxY);
44
45     void update_players(const std::vector<std::string>& names);
46     int get_player_id(const std::string& player);
47
48     void finish_winner();
49     void finish_loser();
50
51     bool is_winner();
52     bool is_loser();
53
54     void clear();
55
56     void update_territory(const int &id, const TeamEnum &team, const int &x,
57                           const int &y);
58
59     std::map<int, std::pair<TeamEnum, std::pair<unsigned int, unsigned int>>>
60     getFlags();
61
62     int get_flag_at(int x, int y);
63
64 };
65
66

```

jun 27, 17 14:46

## MapMonitor.h

Page 2/2

```
67 #endif //Z_TPGRUPAL_MAPMONITOR_H
```

jun 27, 17 14:46

## MapMonitor.cpp

Page 1/2

```
1 #include "MapMonitor.h"
2 #include <Lock.h>
3
4
5 void MapMonitor::setCell(unsigned int xCoordinate, unsigned int yCoordinate,
6                          std::string terrainType) {
7     Lock l(m);
8     map.setCell(xCoordinate, yCoordinate, terrainType);
9 }
10
11 void MapMonitor::initializeMap(unsigned int xSize, unsigned int ySize) {
12     Lock l(m);
13     map.initializeMap(xSize, ySize);
14 }
15
16 unsigned int MapMonitor::getXSize() {
17     Lock l(m);
18     return map.getXSize();
19 }
20
21 unsigned int MapMonitor::getYSize() {
22     Lock l(m);
23     return map.getYSize();
24 }
25
26 std::string MapMonitor::getTerrainTypeAt(unsigned int xCoordinate,
27                                          unsigned int yCoordinate) {
28     Lock l(m);
29     return map.getTerrainTypeAt(xCoordinate, yCoordinate);
30 }
31
32 void MapMonitor::markAsSelectedInRange(bool& terrainSelected, double
33                                       xStartCoordinate,
34                                       double yStartCoordinate,
35                                       double xFinishCoordinate,
36                                       double yFinishCoordinate) {
37     //todo implementar esto que falta ver el tema de accidentales sobre el mapa.
38 }
39
40 void MapMonitor::addNature(Nature nature) {
41     Lock l(m);
42     map.addNature(nature);
43 }
44
45 std::vector<Nature>
46 MapMonitor::getNatureToDraw(unsigned int minX, unsigned int maxX,
47                             unsigned int minY, unsigned int maxY) {
48     Lock l(m);
49     std::vector<Nature> returnVector;
50
51     for (Nature &nature : map.getNature()) {
52         if (nature.getPosition().first >= minX and
53             nature.getPosition().first <= maxX and
54             nature.getPosition().second >= minY and
55             nature.getPosition().second <= maxY) {
56             returnVector.emplace_back(nature);
57         }
58     }
59     return returnVector;
60 }
61
62 void MapMonitor::update_players(const std::vector<std::string> &names) {
63     Lock l(m);
64     this->players = names;
65 }
66
```

jun 27, 17 14:46

## MapMonitor.cpp

Page 2/2

```

67 int MapMonitor::get_player_id(const std::string &player) {
68     int id = 1; // id 0 is neutral
69     Lock l(m);
70     auto it = players.begin();
71     for (; it != players.end(); ++it) {
72         if (*it == player) {
73             break;
74         }
75         id++;
76     }
77     if (it == players.end()) {
78         return 0;
79     }
80     return id; // Not found, return NEUTRAL id
81 }
82
83 void MapMonitor::finish_winner() {
84     Lock l(m);
85     winner = true;
86 }
87
88 void MapMonitor::finish_loser() {
89     Lock l(m);
90     loser = true;
91 }
92
93 bool MapMonitor::is_winner() {
94     Lock l(m);
95     return winner;
96 }
97
98 bool MapMonitor::is_loser() {
99     Lock l(m);
100    return loser;
101 }
102
103
104 void MapMonitor::clear() {
105     Lock l(m);
106     loser = false;
107     winner = false;
108     map.clear();
109     players.clear();
110 }
111
112
113 void MapMonitor::update_territory(const int &id, const TeamEnum &team,
114                                  const int &x, const int &y) {
115     Lock l(m);
116     map.update_territory(id, team, x, y);
117 }
118
119 std::map<int, std::pair<TeamEnum, std::pair<unsigned int, unsigned int>>>
120 MapMonitor::getFlags() {
121     Lock l(m);
122     return map.getFlags();
123 }
124
125 int MapMonitor::get_flag_at(int x, int y) {
126     Lock l(m);
127     return map.get_flag_at(x, y);
128 }

```

jun 27, 17 14:46

## Map.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_MAP_H
2  #define Z_TPGRUPAL_MAP_H
3
4
5  #include <vector>
6  #include "Cell.h"
7  #include "Nature.h"
8  #include "enums/TeamEnum.h"
9  #include <string>
10 #include <map>
11
12 class Map {
13 private:
14     std::vector<std::vector<Cell>> baseMap;
15     std::vector<Nature> nature;
16     /**vector storing flags' positions.
17      * Key = ID,
18      * Value = pair team, coordinates
19      */
20     std::map<int, std::pair<TeamEnum, std::pair<unsigned int, unsigned int>>>
21         flags;
22
23 public:
24     void initializeMap(unsigned int xSize, unsigned int ySize);
25
26     void addNature(Nature nature);
27
28     /* vector storing all nature, i.e. rocks, and all which is not plain
29      * terrain or buildings and independent units, in the map. */
30     std::vector<Nature> getNature();
31
32     void setCell(unsigned int xCoordinate,
33                 unsigned int yCoordinate,
34                 std::string terrainType);
35
36     unsigned int getXSize();
37
38     unsigned int getYSize();
39
40     std::string getTerrainTypeAt(unsigned int xCoordinate,
41                                 unsigned int yCoordinate);
42
43
44     void clear();
45
46     void update_territory(const int &id, const TeamEnum &team, const int &x,
47                         const int &y);
48
49     std::map<int, std::pair<TeamEnum, std::pair<unsigned int, unsigned int>>>
50     getFlags();
51
52     int get_flag_at(int x, int y);
53 };
54
55
56 #endif //Z_TPGRUPAL_MAP_H

```



jun 27, 17 14:46

Map.cpp

Page 1/2

```

1  #include "Map.h"
2
3  #define LENIENCY 3
4
5  void Map::initializeMap(unsigned int xSize, unsigned int ySize) {
6      baseMap.resize(xSize);
7      for (int i = 0; i < xSize; ++i) {
8          baseMap[i].resize(ySize);
9      }
10 }
11
12 void Map::setCell(unsigned int xCoordinate, unsigned int yCoordinate,
13                  std::string terrainType) {
14     baseMap.at(xCoordinate).at(yCoordinate).assignTerrainType(terrainType);
15 }
16
17 unsigned int Map::getXSize() {
18     return (unsigned int) baseMap.size();
19 }
20
21 unsigned int Map::getYSize() {
22     if (!baseMap.size()) {
23         return 0;
24     }
25     return (unsigned int) baseMap[0].size();
26 }
27
28 std::string
29 Map::getTerrainTypeAt(unsigned int xCoordinate, unsigned int yCoordinate) {
30     return baseMap.at(xCoordinate).at(yCoordinate).getTerrainType();
31 }
32
33 void Map::addNature(Nature nature) {
34     this->nature.emplace_back(nature);
35 }
36
37 std::vector<Nature> Map::getNature() {
38     return nature;
39 }
40
41 void Map::clear() {
42     baseMap.clear();
43     nature.clear();
44     flags.clear();
45 }
46
47 void Map::update_territory(const int &id, const TeamEnum &team, const int &x,
48                           const int &y) {
49     flags[id] = {team, {x, y}};
50 }
51
52 std::map<int, std::pair<TeamEnum, std::pair<unsigned int, unsigned int>>>
53 Map::getFlags() {
54     return flags;
55 }
56
57 int Map::get_flag_at(int x, int y) {
58     for (auto flag : flags) {
59         std::pair<int, int> position = flag.second.second;
60         int x_abs = abs(x - position.first);
61         int y_abs = abs(y - position.second);
62         if (x_abs ≤ LENIENCY ^ y_abs ≤ LENIENCY) {
63             return flag.first;
64         }
65     }
66     return -1;

```

jun 27, 17 14:46

Map.cpp

Page 2/2

```

67 }

```

jun 27, 17 14:46

main.cpp

Page 1/1

```

1  #include <gtkmm.h>
2  #include <iostream>
3  #include "ClientThread.h"
4  #include "GameBuilder.h"
5  #include "Game.h"
6  #include <split.h>
7
8  #define SUCCESSRETURNCODE 0
9  #define ERRORCODE 1
10 int main(int argc, char **argv) {
11     try {
12         auto app = Gtk::Application::create();
13
14         GameBuilder builder;
15         InitialWindow *window = builder.get_initial_window();
16         app->run(*window);
17
18         // Once initial window closes, we fetch the socket
19         std::shared_ptr<ServerMessenger> m = window->get_socket();
20         std::string player_name = window->get_username();
21         if (m) {
22             ServerMessenger messenger = *m.get();
23             MapMonitor mapMonitor;
24             UnitsMonitor units_monitor;
25             BuildingsMonitor buildingsMonitor;
26             ClientThread clientThread(units_monitor, buildingsMonitor,
27                                     mapMonitor, messenger, builder);
28             clientThread.start();
29             bool keep_playing = true;
30             while(keep_playing) {
31                 // Starts the game
32                 Game g(builder, messenger, player_name, mapMonitor,
33                     units_monitor, buildingsMonitor);
34
35                 // Game finishes
36                 keep_playing = g.get_play_again_status();
37             }
38             clientThread.finish();
39             clientThread.join();
40         }
41
42         return SUCCESSRETURNCODE;
43     } catch (std::exception const &ex) {
44         std::cerr << ex.what() << std::endl;
45         return ERRORCODE;
46     }
47 }

```

jun 27, 17 14:46

Game.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_GAME_H
2  #define Z_TPGRUPAL_GAME_H
3
4
5  #include "GameBuilder.h"
6  #include "ServerMessenger.h"
7  #include "windows/ResultWindow.h"
8
9  class Game {
10     std::string me;
11
12     ServerMessenger &messenger;
13     MapMonitor& mapMonitor;
14     UnitsMonitor& units_monitor;
15     BuildingsMonitor& buildingsMonitor;
16
17     // Windows of different game stages
18     MenuWindow* menu;
19     LobbyWindow* lobby;
20     GameWindow* game;
21     ResultWindow* result;
22
23     bool play_again = false;
24 public:
25     Game(GameBuilder& builder, ServerMessenger& server_messenger,
26         const std::string& player_name, MapMonitor& map,
27         UnitsMonitor& units, BuildingsMonitor& buildings);
28
29     bool get_play_again_status();
30
31     ~Game();
32 private:
33     void start_menu();
34
35     void start_lobby() const;
36
37     void start_game(const std::vector<std::string> &names);
38
39     void results_screen(bool winner, bool loser);
40 };
41
42
43 #endif //Z_TPGRUPAL_GAME_H

```

jun 27, 17 14:46

## Game.cpp

Page 1/2

```

1  #include "Game.h"
2  #include "ClientThread.h"
3
4  void Game::start_game(const std::vector<std::string> &names) {
5      game->update_name(me);
6      // Start up the game
7      game->setResources(&units_monitor, &buildingsMonitor,
8                      &mapMonitor, &messenger, me);
9
10     auto app = Gtk::Application::create();
11     app->run(*game);
12 }
13
14 Game::Game(GameBuilder &builder, ServerMessenger &server_messenger,
15           const std::string& player_name, MapMonitor& map,
16           UnitsMonitor& units, BuildingsMonitor& buildings) :
17     messenger(server_messenger),
18     me(player_name),
19     menu(builder.get_menu_window()),
20     lobby(builder.get_lobby_window()),
21     game(builder.get_window()),
22     result(builder.get_result_window()),
23     mapMonitor(map),
24     units_monitor(units),
25     buildingsMonitor(buildings)
26 {
27
28     start_menu();
29
30     if (menu->joined_lobby()) {
31         start_lobby();
32         if (lobby->game_started()) {
33             std::vector<std::string> names = lobby->get_player_names();
34             mapMonitor.update_players(names);
35             start_game(names);
36         }
37     }
38
39     bool winner = mapMonitor.is_winner();
40     bool loser = mapMonitor.is_loser();
41     results_screen(winner, loser);
42 }
43
44 void Game::start_lobby() const {
45     lobby->set_messenger(messenger);
46     auto app = Gtk::Application::create();
47     app->run(*lobby);
48 }
49
50 void Game::start_menu() {
51     menu->load_messenger(&messenger);
52     messenger.send("lobbyinfo");
53     auto app = Gtk::Application::create();
54     app->run(*menu);
55 }
56
57 void Game::results_screen(bool winner, bool loser) {
58     if (!winner ^ !loser) { // Played closed the window before game was over
59         play_again = false;
60         return;
61     } else if (winner) {
62         result->display_win_screen();
63     } else {
64         result->display_lose_screen();
65     }
66 }

```

jun 27, 17 14:46

## Game.cpp

Page 2/2

```

67     auto app = Gtk::Application::create();
68     app->run(*result);
69     bool play_again = result->go_back_to_menu();
70     if (play_again) {
71         messenger.send("returntomenu");
72     }
73     this->play_again = play_again;
74 }
75
76 bool Game::get_play_again_status() {
77     return play_again;
78 }
79
80 Game::~Game() { // Game finishes, clear assets
81     units_monitor.clear();
82     mapMonitor.clear();
83     buildingsMonitor.clear();
84 }

```

jun 27, 17 14:46

## GameBuilder.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_GAMEBUILDER_H
2  #define Z_TPGRUPAL_GAMEBUILDER_H
3
4  #include <gtkmm.h>
5  #include <string>
6
7  #include "windows/GameWindow.h"
8  #include "BuildingsMonitor.h"
9  #include "MapMonitor.h"
10 #include "ServerMessenger.h"
11 #include "windows/InitialWindow.h"
12 #include "windows/LobbyWindow.h"
13 #include "windows/MenuWindow.h"
14 #include "windows/ResultWindow.h"
15
16 class GameBuilder {
17     InitialWindow *init_window;
18     MenuWindow* menu_window;
19     LobbyWindow* lobby_window;
20     GameWindow *window;
21     ResultWindow* result_window;
22     Glib::RefPtr<Gtk::Builder> refBuilder;
23
24 public:
25     GameBuilder();
26     ~GameBuilder();
27
28     // returns the generated window
29     GameWindow *get_window();
30
31     InitialWindow *get_initial_window();
32     LobbyWindow* get_lobby_window();
33     MenuWindow* get_menu_window();
34     ResultWindow* get_result_window();
35
36 private:
37     void start();
38     void clean();
39 };
40
41 #endif //Z_TPGRUPAL_GAMEWINDOW_H

```

jun 27, 17 14:46

## GameBuilder.cpp

Page 1/2

```

1  #include <iostream>
2  #include "GameBuilder.h"
3  #include "windows/ResultWindow.h"
4
5
6  GameBuilder::GameBuilder() :
7      init_window(nullptr),
8      menu_window(nullptr),
9      lobby_window(nullptr),
10     window(nullptr),
11     result_window(nullptr)
12 {
13     //Load the GtkBuilder file and instantiate its widgets:
14     start();
15 }
16
17 void GameBuilder::start() {
18     this->refBuilder = Gtk::Builder::create();
19     try {
20         this->refBuilder->add_from_file("Z.glade");
21     }
22     catch (const Glib::FileError &ex) {
23         std::cerr << "FileError: " << ex.what() << std::endl;
24         return;
25     }
26     catch (const Glib::MarkupError &ex) {
27         std::cerr << "MarkupError: " << ex.what() << std::endl;
28         return;
29     }
30     catch (const Gtk::BuilderError &ex) {
31         std::cerr << "BuilderError: " << ex.what() << std::endl;
32         return;
33     }
34 }
35
36 // Save the widget refs in the class attributes
37 this->refBuilder->get_widget_derived("GameWindow", this->window);
38 this->refBuilder->get_widget_derived("InitialWindow", this->init_window);
39 this->refBuilder->get_widget_derived("LobbyWindow", this->lobby_window);
40 this->refBuilder->get_widget_derived("MenuWindow", this->menu_window);
41 this->refBuilder->get_widget_derived("ResultWindow", this->result_window);
42 }
43
44 GameBuilder::~GameBuilder() {
45     clean();
46 }
47
48 GameWindow *GameBuilder::get_window() {
49     return window;
50 }
51
52 InitialWindow *GameBuilder::get_initial_window() {
53     return init_window;
54 }
55
56 LobbyWindow *GameBuilder::get_lobby_window() {
57     return lobby_window;
58 }
59
60 MenuWindow *GameBuilder::get_menu_window() {
61     return menu_window;
62 }
63
64 ResultWindow *GameBuilder::get_result_window() {
65     return result_window;
66 }

```

jun 27, 17 14:46

**GameBuilder.cpp**

Page 2/2

```

67 }
68
69 void GameBuilder::clean() {
70     if (window) {
71         delete window;
72     }
73     if (init_window) {
74         delete init_window;
75     }
76     if (lobby_window) {
77         delete lobby_window;
78     }
79     if (menu_window) {
80         delete menu_window;
81     }
82     if (result_window) {
83         delete result_window;
84     }
85 }
86
87
88

```

jun 27, 17 14:46

**GameArea.h**

Page 1/6

```

1  #ifndef Z_TPGRUPAL_GAMEAREA_H
2  #define Z_TPGRUPAL_GAMEAREA_H
3
4
5  #include <gtkmm/drawingarea.h>
6  #include <utility>
7  #include "BuildingsMonitor.h"
8  #include "MapMonitor.h"
9  #include "Camera.h"
10 #include "enums/TeamEnum.h"
11 #include "enums/ActionsEnum.h"
12 #include "enums/UnitsEnum.h"
13 #include "enums/BuildingsEnum.h"
14 #include "enums/RotationsEnum.h"
15 #include "Counter.h"
16 #include "UnitsMonitor.h"
17 #include <map>
18 #include <string>
19 #include <vector>
20
21 class GameArea : public Gtk::DrawingArea {
22 private:
23     /* shared resources */
24     UnitsMonitor *unitsMonitor;
25     BuildingsMonitor *buildingsMonitor;
26     MapMonitor *mapMonitor;
27
28     std::string owner;
29
30     Camera camera;
31
32     /* general resources */
33     std::map<std::string, Glib::RefPtr<Gdk::Pixbuf>> tiles;
34
35     std::map<NatureEnum, Glib::RefPtr<Gdk::Pixbuf>> nature;
36     Glib::RefPtr<Gdk::Pixbuf> someImg;
37
38     /* map holding all units imgs */
39     std::map<TeamEnum,
40             std::map<UnitsEnum,
41                     std::map<ActionsEnum,
42                             std::map<RotationsEnum,
43                                     std::vector<Glib::RefPtr<Gdk::Pixbuf>>>>>
44             unitsAnimations;
45
46     std::map<RotationsEnum, std::vector<Glib::RefPtr<Gdk::Pixbuf>>> jeepTires;
47
48     /* BUILDINGS RESOURCES */
49     std::map<BuildingsEnum, std::vector<Glib::RefPtr<Gdk::Pixbuf>>> buildings;
50
51     /* map holding all flags */
52     std::map<TeamEnum, std::vector<Glib::RefPtr<Gdk::Pixbuf>>> flags;
53
54     /* declare counter used to know which of the flag imgs
55      * which compose the flag's animation should be showed. This counters are
56      * updated every time on_draw() is called.
57      */
58     Counter flagCounter;
59     Counter standingRobotCounter;
60     Counter walkingRobotCounter;
61     Counter shootingRobotCounter;
62     Counter jeepCounter;
63     Counter tireCounter;
64     Counter tankCounter;
65     Counter mmlCounter;
66     Counter buildingsCounter;

```

jun 27, 17 14:46	GameArea.h	Page 2/6
67	bool move_cmd = false;	
68	std::pair<int, int> coords;	
69		
70		
71	<i>/* DRAWING METHODS */</i>	
72	void drawBaseMap(const Cairo::RefPtr<Cairo::Context> &cr);	
73		
74	void drawTileAt(const Cairo::RefPtr<Cairo::Context> &cr,	
75	unsigned int xTileCoordinate, unsigned int yTileCoordinate,	
76	std::string terrainType);	
77		
78	void drawBuildingsInView(const Cairo::RefPtr<Cairo::Context> &cr);	
79		
80		
81	<i>/* Event handling methods */</i>	
82	bool on_key_press_event(GdkEventKey *event) override;	
83		
84	bool on_button_press_event(GdkEventButton *event) override;	
85		
86	bool on_button_release_event(GdkEventButton *event) override;	
87		
88	<i>/* vars. for event handling */</i>	
89	<i>/* this coordinates are sytem coordinatess */</i>	
90	gdouble xStartCoordinate;	
91	gdouble xFinishCoordinate;	
92	gdouble yStartCoordinate;	
93	gdouble yFinishCoordinate;	
94	bool selectionMade;	
95		
96	<i>/* unitsSelected is true if the players' units are selected. This is used</i>	
97	<i>  to manage user clicks.</i>	
98	<i>  * Turns true in Unit::markAsSelectedInRange, when some unit has been</i>	
99	<i>  * selected.</i>	
100	<i>  * Turns to false at the end of GameArea::processSelection(), when the</i>	
101	<i>  * selection has already been processed. */</i>	
102	bool unitsSelected;	
103	bool buildingSelected;	
104	bool terrainSelected;	
105		
106	void makeSelection();	
107		
108	<i>/* FLAG res loading */</i>	
109	void loadFlagAnimations();	
110		
111	<i>/* UNIT FIRE res loading */</i>	
112	void loadUnitsResources();	
113		
114	void loadGruntFireAnimations();	
115		
116	void loadLaserFireAnimations();	
117		
118	void loadPsychoFireAnimations();	
119		
120	void loadPyroFireAnimations();	
121		
122	void loadSniperFireAnimations();	
123		
124	void loadToughFireAnimations();	
125		
126	void loadBlueGruntFireAnimations();	
127		
128	void loadGreenGruntFireAnimations();	
129		
130	void loadRedGruntFireAnimations();	
131		
132	void loadYellowGruntFireAnimations();	

jun 27, 17 14:46	GameArea.h	Page 3/6
133	void loadBlueLaserFireAnimations();	
134		
135	void loadGreenLaserFireAnimations();	
136		
137	void loadRedLaserFireAnimations();	
138		
139	void loadYellowLaserFireAnimations();	
140		
141	void loadBluePsychoFireAnimations();	
142		
143	void loadGreenPsychoFireAnimations();	
144		
145	void loadRedPsychoFireAnimations();	
146		
147	void loadYellowPsychoFireAnimations();	
148		
149	void loadBluePyroFireAnimation();	
150		
151	void loadGreenPyroFireAnimation();	
152		
153	void loadRedPyroFireAnimation();	
154		
155	void loadYellowPyroFireAnimation();	
156		
157	void loadBlueSniperFireAnimations();	
158		
159	void loadGreenSniperFireAnimations();	
160		
161	void loadRedSniperFireAnimations();	
162		
163	void loadYellowSniperFireAnimations();	
164		
165	void loadBlueToughFireAnimations();	
166		
167	void loadGreenToughFireAnimations();	
168		
169	void loadRedToughFireAnimations();	
170		
171	void loadYellowToughFireAnimations();	
172		
173	<i>/* UNIT WALK res loading */</i>	
174	void loadBlueWalkingAnimations();	
175		
176	void loadGreenWalkingAnimations();	
177		
178	void loadRedWalkingAnimations();	
179		
180	void loadYellowWalkingAnimations();	
181		
182	<i>/* UNIT STAND res loading */</i>	
183	void loadBlueStandingAnimations();	
184		
185	void loadGreenStandingAnimations();	
186		
187	void loadRedStandingAnimations();	
188		
189	void loadYellowStandingAnimations();	
190		
191	<i>/* BUILDING res loading */</i>	
192	void loadBuildingsResources();	
193		
194	<i>/* VEHICLES res loading */</i>	
195	void loadTiresAnimations();	
196		
197	void loadJeepTires();	
198		

jun 27, 17 14:46	GameArea.h	Page 4/6
199	<code>void loadNeuterVehiclesAnimations();</code>	
200		
201	<code>void loadNeuterJeepAnimations();</code>	
202		
203	<code>void loadBlueVehiclesAnimations();</code>	
204		
205	<code>void loadGreenVehiclesAnimations();</code>	
206		
207	<code>void loadRedVehiclesAnimations();</code>	
208		
209	<code>void loadYellowVehiclesAnimations();</code>	
210		
211	<code>void loadBlueJeepAnimations();</code>	
212		
213	<code>void loadBlueLightTankAnimations();</code>	
214		
215	<code>void loadBlueMediumTankAnimations();</code>	
216		
217	<code>void loadBlueMMLAnimations();</code>	
218		
219	<code>void loadBlueHeavyTankAnimations();</code>	
220		
221	<code>void loadGreenJeepAnimations();</code>	
222		
223	<code>void loadGreenLightTankAnimations();</code>	
224		
225	<code>void loadGreenMediumTankAnimations();</code>	
226		
227	<code>void loadGreenMMLAnimations();</code>	
228		
229	<code>void loadGreenHeavyTankAnimations();</code>	
230		
231	<code>void loadRedJeepAnimations();</code>	
232		
233	<code>void loadRedLightTankAnimations();</code>	
234		
235	<code>void loadRedMediumTankAnimations();</code>	
236		
237	<code>void loadRedMMLAnimations();</code>	
238		
239	<code>void loadRedHeavyTankAnimations();</code>	
240		
241	<code>void loadYellowJeepAnimations();</code>	
242		
243	<code>void loadYellowLightTankAnimations();</code>	
244		
245	<code>void loadYellowMediumTankAnimations();</code>	
246		
247	<code>void loadYellowMMLAnimations();</code>	
248		
249	<code>void loadYellowHeavyTankAnimations();</code>	
250		
251	<code>void drawJeepTires(const Cairo::RefPtr&lt;Cairo::Context&gt; &amp;cr,</code>	
252	<code>unsigned int xGraphicCoordinate,</code>	
253	<code>unsigned int yGraphicCoordinate,</code>	
254	<code>RotationsEnum rotation);</code>	
255		
256	<code>void drawUnit(TeamEnum team, UnitsEnum unitType, ActionsEnum actionType,</code>	
257	<code>RotationsEnum rotation, unsigned short unitCounter,</code>	
258	<code>const Cairo::RefPtr&lt;Cairo::Context&gt; &amp;cr,</code>	
259	<code>unsigned int xGraphicCoordinate,</code>	
260	<code>unsigned int yGraphicCoordinate);</code>	
261		
262	<code>void loadResources();</code>	
263		
264		

jun 27, 17 14:46	GameArea.h	Page 5/6
265	<code>/**</code>	
266	<code> * draws all units that are in camera's scope</code>	
267	<code> * @param cr receive smart pointer to cairo context</code>	
268	<code> */</code>	
269	<code>void drawUnitsInMap(const Cairo::RefPtr&lt;Cairo::Context&gt; &amp;cr);</code>	
270		
271	<code>/**</code>	
272	<code> * counters used to know whic img of each animation should be drawn are</code>	
273	<code> * updated. This method should get called once per frame, otherwise it is</code>	
274	<code> * not guaranteed that animations will be correctly drawn.</code>	
275	<code> */</code>	
276	<code>void updateCounters();</code>	
277		
278	<code>protected:</code>	
279	<code>bool on_draw(const Cairo::RefPtr&lt;Cairo::Context&gt; &amp;cr) override;</code>	
280		
281	<code>public:</code>	
282	<code>virtual ~GameArea();</code>	
283		
284	<code>GameArea(BaseObjectType *cobject,</code>	
285	<code>const Glib::RefPtr&lt;Gtk::Builder&gt; &amp;builder);</code>	
286		
287	<code>/**</code>	
288	<code> * initialize shared resources.</code>	
289	<code> */</code>	
290	<code>void</code>	
291	<code>setResources(UnitsMonitor *playersMonitor,</code>	
292	<code>BuildingsMonitor *buildingsMonitor,</code>	
293	<code>MapMonitor *mapMonitor, std::string owner);</code>	
294		
295	<code>void processClick();</code>	
296		
297	<code>void initializeCounters();</code>	
298		
299	<code>unsigned short getCounter(Unit &amp;unit) const;</code>	
300		
301	<code>void processUnitToDrawEnums(UnitsEnum &amp;unitType, ActionsEnum &amp;actionType,</code>	
302	<code>RotationsEnum &amp;rotation) const;</code>	
303		
304	<code>unsigned int cameraToRealMapX(unsigned int coordinate);</code>	
305		
306	<code>unsigned int cameraToRealMapY(unsigned int coordinate);</code>	
307		
308	<code>std::pair&lt;int, int&gt; get_coords();</code>	
309		
310	<code>unsigned int screenMapToCameraX(gdouble coordinate);</code>	
311		
312	<code>unsigned int screenMapToCameraY(gdouble coordinate);</code>	
313		
314	<code>void setMapData();</code>	
315		
316	<code>void drawBuilding(BuildingsEnum buildingType, unsigned short counter,</code>	
317	<code>TeamEnum team,</code>	
318	<code>const Cairo::RefPtr&lt;Cairo::Context&gt; &amp;cr,</code>	
319	<code>unsigned int xGraphicCoordinate,</code>	
320	<code>unsigned int yGraphicCoordinate);</code>	
321		
322	<code>bool unitIsRobot(UnitsEnum unitType);</code>	
323		
324	<code>void drawFlag(const TeamEnum &amp;team, const Cairo::RefPtr&lt;Cairo::Context&gt; &amp;cr,</code>	
325	<code>unsigned int xGraphicCoordinate,</code>	
326	<code>unsigned int yGraphicCoordinate) const;</code>	
327		
328	<code>void drawTerritoriesFlagsInView(const Cairo::RefPtr&lt;Cairo::Context&gt; &amp;cr);</code>	
329		
330	<code>bool unit_selected();</code>	

jun 27, 17 14:46

**GameArea.h**

Page 6/6

```

331
332     bool buildings_selected();
333
334     void drawNatureInView(const Cairo::RefPtr<Cairo::Context> &cr);
335
336     void
337     drawNature(NatureEnum natureType, const Cairo::RefPtr<Cairo::Context> &cr,
338               unsigned int x, unsigned int y);
339
340     void loadMapResources();
341 };
342
343
344 #endif //Z_TPGRUPAL_GAMEAREA_H

```

jun 27, 17 14:46

**GameArea.cpp**

Page 1/11

```

1  #include <gtkmm/builder.h>
2  #include <gdkmm.h>
3  #include <iostream>
4  #include "GameArea.h"
5  #include <giomm.h>
6
7  #define TILESIZE 16      //tile width in pixels. This define is also present
8                          //in AddBuilding Command
9  #define NUMBER_OF_TILES_TO_SHOW 10
10
11  GameArea::GameArea(BaseObjectType *cobject,
12                    const Glib::RefPtr<Gtk::Builder> &builder) :
13      Gtk::DrawingArea(cobject),
14      unitsMonitor(nullptr),
15      buildingsMonitor(nullptr),
16      mapMonitor(nullptr),
17      unitsSelected(false),
18      buildingSelected(false),
19      coords({-1, -1}),
20      /* camera is initialized with size 0,0 because we don't
21       * have this data yet */
22      camera(TILESIZE, 0, 0, NUMBER_OF_TILES_TO_SHOW) {
23      loadResources();
24      initializeCounters();
25
26      add_events(Gdk::EventMask::BUTTON_PRESS_MASK);
27      add_events(Gdk::EventMask::BUTTON_RELEASE_MASK);
28      add_events(Gdk::EventMask::KEY_PRESS_MASK);
29      set_can_focus(true);
30  }
31
32  void GameArea::loadResources() {
33      try {
34          /* load flags animations */
35          loadFlagAnimations();
36          /* load units resources */
37          loadUnitsResources();
38          /* load buildings resources */
39          loadBuildingsResources();
40
41          loadMapResources();
42      } catch (Glib::FileError e) {
43          std::cerr << e.what();
44      }
45  }
46
47  void GameArea::loadMapResources() {
48      /* Load tiles */
49      tiles["Tierra"] = Gdk::Pixbuf::create_from_file(
50          "res/assets/tiles/tierra.png");
51      tiles["Agua"] = Gdk::Pixbuf::create_from_file(
52          "res/assets/tiles/agua.png");
53      tiles["Lava"] = Gdk::Pixbuf::create_from_file(
54          "res/assets/tiles/lava.png");
55      /* Load nature items */
56      nature[NatureEnum::ROCK] = Gdk::Pixbuf::create_from_file(
57          "res/assets/nature/rock.png");
58  }
59
60  GameArea::~GameArea() {}
61
62  bool GameArea::on_draw(const Cairo::RefPtr<Cairo::Context> &cr) {
63      drawBaseMap(cr);
64      drawBuildingsInView(cr);
65      drawTerritoriesFlagsInView(cr);
66      drawUnitsInMap(cr);

```



jun 27, 17 14:46

## GameArea.cpp

Page 2/11

```

67     updateCounters();
68     return true;
69 }
70
71 void GameArea::drawBaseMap(const Cairo::RefPtr<Cairo::Context> &cr) {
72     /* check if map is empty */
73     if (mapMonitor->getXSize() == 0 and mapMonitor->getYSize() == 0) {
74         return;
75     }
76
77     /* cameraPosition is given in pixels.
78     * i,j indicate TILES. */
79     for (unsigned int i = 0; i < NUMBER_OF_TILES_TO_SHOW; ++i) {
80         for (unsigned int j = 0; j < NUMBER_OF_TILES_TO_SHOW; ++j) {
81             drawTileAt(cr, i, j, mapMonitor->getTerrainTypeAt(
82                 camera.getPosition().first / TILESIZE -
83                 NUMBER_OF_TILES_TO_SHOW / 2 + i,
84                 camera.getPosition().second / TILESIZE -
85                 NUMBER_OF_TILES_TO_SHOW / 2 + j));
86         }
87     }
88     drawNatureInView(cr);
89 }
90
91 void GameArea::drawNatureInView(const Cairo::RefPtr<Cairo::Context> &cr) {
92     /* pointers (Nature*) are not used here because we are working with a shared
93     * resource. This way, we copy the units we want to draw in a protected way,
94     * and then we can draw without blocking other code. */
95     std::vector<Nature> natureToDraw = mapMonitor->getNatureToDraw(
96         camera.getPosition().first -
97         (NUMBER_OF_TILES_TO_SHOW * TILESIZE) / 2,
98         camera.getPosition().first +
99         (NUMBER_OF_TILES_TO_SHOW * TILESIZE) / 2,
100        camera.getPosition().second -
101        (NUMBER_OF_TILES_TO_SHOW * TILESIZE) / 2,
102        camera.getPosition().second +
103        (NUMBER_OF_TILES_TO_SHOW * TILESIZE) / 2);
104
105     for (auto &nature : natureToDraw) {
106         /* call actual drawing method */
107         drawNature(nature.getType(), cr,
108             cameraToRealMapX(camera.idealMapToCameraXCoordinate(
109                 nature.getPosition().first)),
110             cameraToRealMapY(camera.idealMapToCameraYCoordinate(
111                 nature.getPosition().second)));
112     }
113 }
114
115 void GameArea::drawNature(NatureEnum natureType,
116     const Cairo::RefPtr<Cairo::Context> &cr,
117     unsigned int x, unsigned int y) {
118     cr->save();
119     Gdk::Cairo::set_source_pixbuf(cr, nature[natureType], x, y);
120     cr->rectangle(x, y, nature[natureType]->get_width(),
121         nature[natureType]->get_height());
122     cr->fill();
123     cr->restore();
124 }
125
126 void GameArea::drawTileAt(const Cairo::RefPtr<Cairo::Context> &cr,
127     unsigned int xTileCoordinate,
128     unsigned int yTileCoordinate,
129     std::string terrainType) {
130     cr->save();
131     auto pixbuf = tiles.find(terrainType);
132     if (pixbuf == tiles.end()) {

```

jun 27, 17 14:46

## GameArea.cpp

Page 3/11

```

133     return;
134 }
135 const unsigned int xGraphicCoordinate = xTileCoordinate * get_width() /
136     NUMBER_OF_TILES_TO_SHOW;
137 const unsigned int yGraphicCoordinate = yTileCoordinate * get_height() /
138     NUMBER_OF_TILES_TO_SHOW;
139 Gdk::Cairo::set_source_pixbuf(cr, pixbuf->second,
140     xGraphicCoordinate,
141     yGraphicCoordinate);
142
143 cr->rectangle(xGraphicCoordinate, yGraphicCoordinate,
144     get_width() / NUMBER_OF_TILES_TO_SHOW,
145     get_height() / NUMBER_OF_TILES_TO_SHOW);
146 cr->fill();
147 cr->restore();
148 }
149
150 void GameArea::drawJeepTires(const Cairo::RefPtr<Cairo::Context> &cr,
151     unsigned int xGraphicCoordinate,
152     unsigned int yGraphicCoordinate,
153     RotationsEnum rotation) {
154
155     cr->save();
156     /* first draw jeepTires */
157     Gdk::Cairo::set_source_pixbuf(cr, jeepTires.at(rotation).
158         at(tireCounter.getCounter()),
159         xGraphicCoordinate, yGraphicCoordinate);
160     cr->rectangle(xGraphicCoordinate, yGraphicCoordinate,
161         jeepTires.at(rotation).
162             at(tireCounter.getCounter())->get_width(),
163         jeepTires.at(rotation).
164             at(tireCounter.getCounter())->get_height());
165     cr->fill();
166     cr->restore();
167 }
168
169 void GameArea::drawUnitsInMap(const Cairo::RefPtr<Cairo::Context> &cr) {
170     /* pointers (Unit*) are not used here because we are working with a shared
171     * resource. This way, we copy the units we want to draw in a protected way,
172     * and then we can draw without blocking other code. */
173     std::vector<Unit> unitsToDraw = unitsMonitor->getUnitsToDraw(
174         camera.getPosition().first -
175         (NUMBER_OF_TILES_TO_SHOW * TILESIZE) / 2,
176         camera.getPosition().first +
177         (NUMBER_OF_TILES_TO_SHOW * TILESIZE) / 2,
178         camera.getPosition().second -
179         (NUMBER_OF_TILES_TO_SHOW * TILESIZE) / 2,
180         camera.getPosition().second +
181         (NUMBER_OF_TILES_TO_SHOW * TILESIZE) / 2);
182
183     for (auto &unit : unitsToDraw) {
184         /* check what is being drawn, and choose the counter appropriately. */
185         unsigned short counter;
186         counter = getCounter(unit);
187
188         /* call actual drawing method */
189         drawUnit(unit.getTeam(), unit.getType(), unit.getAction(),
190             unit.getRotation(),
191             counter, cr,
192             cameraToRealMapX(
193                 camera.idealMapToCameraXCoordinate(
194                     unit.getXCoordinate())),
195             cameraToRealMapY(
196                 camera.idealMapToCameraYCoordinate(
197                     unit.getYCoordinate())));
198     }

```

jun 27, 17 14:46

## GameArea.cpp

Page 4/11

```

199 }
200
201 void GameArea::drawUnit(TeamEnum team, UnitsEnum unitType,
202     ActionsEnum actionType,
203     RotationsEnum rotation, unsigned short unitCounter,
204     const Cairo::RefPtr<Cairo::Context> &cr,
205     unsigned int xGraphicCoordinate,
206     unsigned int yGraphicCoordinate) {
207     cr->save();
208     /* adapt given data to saved imgs. Applies to vehicles */
209     if (unitType == UnitsEnum::JEEP ^
210         rotation != RotationsEnum::r090 ^
211         rotation != RotationsEnum::r270) {
212         /* rotations 090 and 270 dont have tires */
213         drawJeepTires(cr, xGraphicCoordinate, yGraphicCoordinate, rotation);
214     }
215     processUnitToDrawEnums(unitType, actionType, rotation);
216
217     auto team_map = unitsAnimations.find(team);
218     if (team_map == unitsAnimations.end()) {
219         std::cerr << "Drawing failed at finding valid team" << std::endl;
220     }
221
222     auto unit_map = team_map->second.find(unitType);
223     if (unit_map == team_map->second.end()) {
224         std::cerr << "Drawing failed at finding valid unitType" << std::endl;
225     }
226
227     auto actions_map = unit_map->second.find(actionType);
228     if (actions_map == unit_map->second.end()) {
229         std::cerr << "Drawing failed at finding valid actionType" << std::endl;
230     }
231
232     auto rotations_map = actions_map->second.find(rotation);
233     if (rotations_map == actions_map->second.end()) {
234         std::cerr << "Drawing failed at finding valid rotation" << std::endl;
235     }
236
237     if (unitIsRobot(unitType)) {
238         cr->scale(1.5, 1.5);
239         xGraphicCoordinate = xGraphicCoordinate/1.5;
240         yGraphicCoordinate = yGraphicCoordinate/1.5;
241     }
242
243     auto next = rotations_map->second.at(unitCounter);
244     /* perform actual drawing */
245     Gdk::Cairo::set_source_pixbuf(cr, next,
246         xGraphicCoordinate, yGraphicCoordinate);
247
248     cr->rectangle(xGraphicCoordinate, yGraphicCoordinate, next->get_width(),
249         next->get_height());
250     cr->fill();
251     cr->restore();
252 }
253
254 void
255 GameArea::processUnitToDrawEnums(UnitsEnum &unitType, ActionsEnum &actionType,
256     RotationsEnum &rotation) const {
257     if (unitType == UnitsEnum::HEAVY_TANK
258         or unitType == UnitsEnum::LIGHT_TANK
259         or unitType == UnitsEnum::MEDIUM_TANK
260         or unitType == UnitsEnum::MML) {
261         actionType = ActionsEnum::STAND;
262         /* same assets are used for given rotations;
263          * e.g.: 135 and 315 are drawn with same img */
264         if (rotation == RotationsEnum::r135) {

```

jun 27, 17 14:46

## GameArea.cpp

Page 5/11

```

265         rotation = RotationsEnum::r315;
266     } else if (rotation == RotationsEnum::r180) {
267         rotation = RotationsEnum::r000;
268     } else if (rotation == RotationsEnum::r225) {
269         rotation = RotationsEnum::r045;
270     } else if (rotation == RotationsEnum::r270) {
271         rotation = RotationsEnum::r090;
272     }
273 } else if ((unitType == UnitsEnum::GRUNT
274     or unitType == UnitsEnum::LASER
275     or unitType == UnitsEnum::PSYCHO
276     or unitType == UnitsEnum::PYRO
277     or unitType == UnitsEnum::SNIPER
278     or unitType == UnitsEnum::TOUGH)
279     and(actionType == ActionsEnum::MOVE
280         or actionType == ActionsEnum::STAND)) {
281     /* because same imgs are used to draw all different types of robots
282      * when these are moving or standing still, if this is the case, we
283      * set the unit type to generic robot */
284     unitType = UnitsEnum::GENERIC_ROBOT;
285 }
286 if (unitType == UnitsEnum::JEEP) {
287     actionType = ActionsEnum::STAND;
288 }
289 }
290
291 void GameArea::drawBuildingsInView(const Cairo::RefPtr<Cairo::Context> &cr) {
292     /* pointers (Unit*) are not used here because we are working with a shared
293      * resource. This way, we copy the units we want to draw in a protected way,
294      * and then we can draw without blocking other code. */
295     std::vector<Building> buildingsToDraw =
296         buildingsMonitor->getBuildingsToDraw(
297             camera.getPosition().first - (NUMBER_OF_TILES_TO_SHOW *
298                 TILESIZEx / 2,
299             camera.getPosition().first + (NUMBER_OF_TILES_TO_SHOW *
300                 TILESIZEx / 2,
301             camera.getPosition().second - (NUMBER_OF_TILES_TO_SHOW *
302                 TILESIZey / 2,
303             camera.getPosition().second + (NUMBER_OF_TILES_TO_SHOW *
304                 TILESIZey / 2));
305
306     for (auto &building : buildingsToDraw) {
307         /* call actual drawing method */
308         drawBuilding(building.getBuildingType(), buildingsCounter.getCounter(),
309             building.getTeam(), cr,
310             cameraToRealMapX(camera.idealMapToCameraXCoordinate
311                 (building.getXCoordinate())),
312             cameraToRealMapY(camera.idealMapToCameraYCoordinate
313                 (building.getYCoordinate())));
314     }
315 }
316
317 void GameArea::drawBuilding(BuildingsEnum buildingType, unsigned short counter,
318     TeamEnum team,
319     const Cairo::RefPtr<Cairo::Context> &cr,
320     unsigned int xGraphicCoordinate,
321     unsigned int yGraphicCoordinate) {
322     cr->save();
323     Glib::RefPtr<Gdk::Pixbuf> next = buildings.at(buildingType).at(counter);
324
325     /* perform actual drawing */
326     Gdk::Cairo::set_source_pixbuf(cr, next,
327         xGraphicCoordinate, yGraphicCoordinate);
328
329     cr->rectangle(xGraphicCoordinate, yGraphicCoordinate, next->get_width(),
330         next->get_height());

```

jun 27, 17 14:46

## GameArea.cpp

Page 6/11

```

331     cr->fill();
332     cr->restore();
333     /* draw flag */
334     if (buildingType != BuildingsEnum::FORT_DESTROYED or
335         buildingType != BuildingsEnum::FORT) {
336         /* forts' flags are territories' flags, so we dont draw them with the
337          * building, but with the other territories' flags instead. */
338         //todo check flag's optimal position
339         drawFlag(team, cr, xGraphicCoordinate, yGraphicCoordinate);
340     }
341 }
342 }
343
344 void GameArea::drawFlag(const TeamEnum &team,
345                        const Cairo::RefPtr<Cairo::Context> &cr,
346                        unsigned int xGraphicCoordinate,
347                        unsigned int yGraphicCoordinate) const {
348     cr->save();
349     Gdk::Cairo::set_source_pixbuf(cr,
350                                  flags.at(team).at(flagCounter.getCounter()),
351                                  xGraphicCoordinate, yGraphicCoordinate);
352
353     cr->rectangle(xGraphicCoordinate,
354                  yGraphicCoordinate,
355                  flags.at(team).at(flagCounter.getCounter())->get_width(),
356                  flags.at(team).at(flagCounter.getCounter())->get_height());
357     cr->fill();
358     cr->restore();
359 }
360
361 void GameArea::drawTerritoriesFlagsInView(const Cairo::RefPtr<Cairo::Context> &c
362 r) {
363     std::map<int, std::pair<TeamEnum, std::pair<unsigned int, unsigned int>>>
364     flagsToDraw = mapMonitor->getFlags();
365     for (auto &flag : flagsToDraw) {
366         /* call actual drawing method */
367         drawFlag(flag.second.first, cr,
368                  cameraToRealMapX(camera.idealMapToCameraXCoordinate(
369                      flag.second.second.first)),
370                  cameraToRealMapY(camera.idealMapToCameraYCoordinate(
371                      flag.second.second.second)));
372     }
373 }
374
375 void GameArea::setResources(UnitsMonitor *unitsMonitor,
376                            BuildingsMonitor *buildingsMonitor,
377                            MapMonitor *mapMonitor, std::string owner) {
378     this->unitsMonitor = unitsMonitor;
379     this->buildingsMonitor = buildingsMonitor;
380     this->mapMonitor = mapMonitor;
381     this->owner = owner;
382 }
383
384 bool GameArea::on_key_press_event(GdkEventKey *event) {
385     if (event->keyval == GDK_KEY_Up and event->keyval == GDK_KEY_Left) {
386         camera.moveUp();
387         camera.moveLeft();
388         //returning true, cancels the propagation of the event
389         return true;
390     } else if (event->keyval == GDK_KEY_Down) {
391         camera.moveDown();
392         //returning true, cancels the propagation of the event
393         return true;
394     } else if (event->keyval == GDK_KEY_Left) {
395         camera.moveLeft();

```

jun 27, 17 14:46

## GameArea.cpp

Page 7/11

```

396         //returning true, cancels the propagation of the event
397         return true;
398     } else if (event->keyval == GDK_KEY_Right) {
399         camera.moveRight();
400         //returning true, cancels the propagation of the event
401         return true;
402     } else if (event->keyval == GDK_KEY_Up) {
403         camera.moveUp();
404         //returning true, cancels the propagation of the event
405         return true;
406     }
407     // todo ver si el event handling se pasa arriba o no
408     //if the event has not been handled, call the base class
409     return Gtk::DrawingArea::on_key_press_event(event);
410 }
411
412 bool GameArea::on_button_press_event(GdkEventButton *event) {
413     /** From https://developer.gnome.org/gdk3/stable/gdk3-Event
414      * -Structures.html#GdkEventButton
415      *
416      * GdkEventType type;      the type of the event (GDK_BUTTON_PRESS,
417      *                          GDK_2BUTTON_PRESS, GDK_3BUTTON_PRESS or
418      *                          GDK_BUTTON_RELEASE).
419      *
420      * GdkWindow *window;      the window which received the event.
421      *
422      * gint8 send_event;      TRUE if the event was sent explicitly.
423      *
424      * guint32 time;          the time of the event in milliseconds.
425      *
426      * gdouble x;             the x coordinate of the pointer relative to the
427      *                          window.
428      *
429      * gdouble y;             the y coordinate of the pointer relative to the
430      *                          window.
431      *
432      * gdouble *axes;        x , y translated to the axes of device , or NULL
433      *                          if device is the mouse.
434      *
435      * guint state;          a bit-mask representing the state of the modifier
436      *                          keys (e.g. Control, Shift and Alt) and the pointer
437      *                          buttons. See GdkModifierType.
438      *
439      * guint button;         the button which was pressed or released, numbered
440      *                          from 1 to 5. Normally button 1 is the left mouse
441      *                          button, 2 is the middle button, and 3 is the right
442      *                          button. On 2-button mice, the middle button can
443      *                          often be simulated by pressing both
444      *                          mouse buttons together.
445      *
446      * GdkDevice *device;    the master device that the event originated from.
447      *                          Use gdk_event_get_source_device()
448      *                          to get the slave device.
449      *
450      * gdouble x_root;       the x coordinate of the pointer relative to
451      *                          the root of the screen.
452      *
453      * gdouble y_root;       the y coordinate of the pointer relative to
454      *                          the root of the screen.
455      *
456      */
457     if (event->button == 1 v event->button == 3) {
458         unitsMonitor->wipeSelected();
459         buildingsMonitor->wipe_selected();
460         unitsSelected = false;
461         buildingSelected = false;

```

jun 27, 17 14:46

## GameArea.cpp

Page 8/11

```

462     xStartCoordinate = event->x;
463     yStartCoordinate = event->y;
464     /* returning true, cancels the propagation of the event */
465 }
466 return true;
467 }
468
469 bool GameArea::on_button_release_event(GdkEventButton *event) {
470     if (event->button == 1 || event->button == 3) {
471         xFinishCoordinate = event->x;
472         yFinishCoordinate = event->y;
473         makeSelection();
474         coords = {camera.cameraToMapXCoordinate(screenMapToCameraX(event->x)),
475                 camera.cameraToMapYCoordinate(screenMapToCameraY(event->y))};
476         /* returning true, cancels the propagation of the event. We return
477          * false, so the event can be handled by the game window
478          */
479     }
480     return false;
481 }
482
483 void GameArea::makeSelection() {
484     /* tell each of the structures storing objects in the map to mark as
485      * selected the items which are within the mouse selection */
486     //todo filter out other players' units.
487     unitsMonitor->markAsSelectedInRange(unitsSelected,
488                                         camera.cameraToMapXCoordinate(
489                                             screenMapToCameraX(
490                                                 xStartCoordinate)),
491                                         camera.cameraToMapYCoordinate(
492                                             screenMapToCameraY(
493                                                 yStartCoordinate)),
494                                         camera.cameraToMapXCoordinate(
495                                             screenMapToCameraX(
496                                                 xFinishCoordinate)),
497                                         camera.cameraToMapYCoordinate(
498                                             screenMapToCameraY(
499                                                 yFinishCoordinate)));
500     if (!unitsSelected) {
501         buildingsMonitor->markAsSelectedInRange(
502             buildingSelected,
503             camera.cameraToMapXCoordinate(
504                 screenMapToCameraX(xStartCoordinate)),
505             camera.cameraToMapYCoordinate(
506                 screenMapToCameraY(yStartCoordinate)),
507             camera.cameraToMapXCoordinate(
508                 screenMapToCameraX(xFinishCoordinate)),
509             camera.cameraToMapYCoordinate(
510                 screenMapToCameraY(yFinishCoordinate)));
511     } else {
512         mapMonitor->markAsSelectedInRange(
513             terrainSelected,
514             camera.cameraToMapXCoordinate(
515                 screenMapToCameraX(xStartCoordinate)),
516             camera.cameraToMapYCoordinate(
517                 screenMapToCameraY(yStartCoordinate)),
518             camera.cameraToMapXCoordinate(
519                 screenMapToCameraX(xFinishCoordinate)),
520             camera.cameraToMapYCoordinate(
521                 screenMapToCameraY(yFinishCoordinate)));
522     }
523     selectionMade = false;
524 }
525
526 void GameArea::processClick() {
527

```

jun 27, 17 14:46

## GameArea.cpp

Page 9/11

```

528     //todo complete method processClick
529     if (unitsSelected) {
530     }
531 }
532
533 void GameArea::loadUnitsResources() {
534     /* load fire animations */
535     loadGruntFireAnimations();
536     loadLaserFireAnimations();
537     loadPsychoFireAnimations();
538     loadPyroFireAnimations();
539     loadSniperFireAnimations();
540     loadToughFireAnimations();
541
542     /* load walking animations */
543     loadBlueWalkingAnimations();
544     loadGreenWalkingAnimations();
545     loadRedWalkingAnimations();
546     loadYellowWalkingAnimations();
547
548     /* load standing animations */
549     loadBlueStandingAnimations();
550     loadGreenStandingAnimations();
551     loadRedStandingAnimations();
552     loadYellowStandingAnimations();
553
554     /* load vehicles' animations */
555     loadTiresAnimations();
556     loadNeuterVehiclesAnimations();
557     loadBlueVehiclesAnimations();
558     loadGreenVehiclesAnimations();
559     loadRedVehiclesAnimations();
560     loadYellowVehiclesAnimations();
561 }
562
563 unsigned short GameArea::getCounter(Unit &unit) const {
564     if (unit.getType() == UnitsEnum::JEEP) {
565         if (unit.getTeam() == TeamEnum::NEUTRAL) {
566             return 0;
567         }
568         return jeepCounter.getCounter();
569     } else if (unit.getType() == UnitsEnum::LIGHT_TANK or
570               unit.getType() == UnitsEnum::MEDIUM_TANK or
571               unit.getType() == UnitsEnum::HEAVY_TANK) {
572         return tankCounter.getCounter();
573     } else if (unit.getType() == UnitsEnum::MML){
574         return mmlCounter.getCounter();
575     } else if (unit.getAction() == ActionsEnum::FIRE) {
576         return shootingRobotCounter.getCounter();
577     } else if (unit.getAction() == ActionsEnum::MOVE) {
578         return walkingRobotCounter.getCounter();
579     } else {
580         return standingRobotCounter.getCounter();
581     }
582 }
583
584 void GameArea::updateCounters() {
585     /* update units counters */
586     flagCounter.updateCounter();
587
588     shootingRobotCounter.updateCounter();
589
590     walkingRobotCounter.updateCounter();
591
592     standingRobotCounter.updateCounter();
593

```

jun 27, 17 14:46

## GameArea.cpp

Page 10/11

```

594     jeepCounter.updateCounter();
595
596     tireCounter.updateCounter();
597
598     tankCounter.updateCounter();
599
600     mmlCounter.updateCounter();
601     /* end update counter section */
602 }
603
604 void GameArea::initializeCounters() {
605     /* one of the vectors of each category is accessed to get the size of the
606      * vectors of all the category. This is possible because all vectors of
607      * the same category share the same size */
608     flagCounter.initialize(flags.at(TeamEnum::BLUE).size());
609
610     jeepCounter.initialize(unitsAnimations.operator[](TeamEnum::BLUE)
611                           [UnitsEnum::JEEP][ActionsEnum::STAND]
612                           [RotationsEnum::r000].size());
613
614     tireCounter.initialize(jeepTires.at(RotationsEnum::r000).size());
615
616     standingRobotCounter.initialize(unitsAnimations.operator[](TeamEnum::BLUE)
617                                    [UnitsEnum::GENERIC_ROBOT]
618                                    [ActionsEnum::STAND]
619                                    [RotationsEnum::r000].size());
620
621     walkingRobotCounter.initialize(unitsAnimations.operator[](TeamEnum::BLUE)
622                                   [UnitsEnum::GENERIC_ROBOT][ActionsEnum::MOVE]
623                                   [RotationsEnum::r000].size());
624
625     shootingRobotCounter.initialize(unitsAnimations.operator[](TeamEnum::BLUE)
626                                    [UnitsEnum::PSYCHO][ActionsEnum::FIRE]
627                                    [RotationsEnum::r000].size());
628
629     tankCounter.initialize(unitsAnimations.operator[](TeamEnum::BLUE)
630                            [UnitsEnum::LIGHT_TANK][ActionsEnum::STAND]
631                            [RotationsEnum::r000].size());
632
633     mmlCounter.initialize(unitsAnimations.operator[](TeamEnum::BLUE)
634                           [UnitsEnum::MML][ActionsEnum::STAND]
635                           [RotationsEnum::r000].size());
636
637     buildingsCounter.initialize(buildings.at(BuildingsEnum::FORT).size());
638 }
639
640 unsigned int GameArea::cameraToRealMapX(unsigned int coordinate) {
641     return get_width() * coordinate / (NUMBER_OF_TILES_TO_SHOW * TILESIZE);
642 }
643
644 unsigned int GameArea::cameraToRealMapY(unsigned int coordinate) {
645     return get_height() * coordinate / (NUMBER_OF_TILES_TO_SHOW * TILESIZE);
646 }
647
648 std::pair<int, int> GameArea::get_coords() {
649     return coords;
650 }
651
652 unsigned int GameArea::screenMapToCameraX(gdouble coordinate) {
653     return (NUMBER_OF_TILES_TO_SHOW * TILESIZE * coordinate) / (get_width());
654 }
655
656 unsigned int GameArea::screenMapToCameraY(gdouble coordinate) {
657     return (NUMBER_OF_TILES_TO_SHOW * TILESIZE * coordinate) / (get_height());
658 }
659

```

jun 27, 17 14:46

## GameArea.cpp

Page 11/11

```

660 void GameArea::setMapData() {
661     this->camera.setMapWidth(mapMonitor->getXSize());
662     this->camera.setMapHeight(mapMonitor->getYSize());
663 }
664
665 bool GameArea::unitIsRobot(UnitsEnum unitType) {
666     return (unitType == UnitsEnum::GENERIC_ROBOT or
667            unitType == UnitsEnum::GRUNT or unitType == UnitsEnum::LASER or
668            unitType == UnitsEnum::PSYCHO or unitType == UnitsEnum::PYRO or
669            unitType == UnitsEnum::SNIPER or unitType == UnitsEnum::TOUGH);
670 }
671
672 bool GameArea::unit_selected() {
673     return unitsSelected;
674 }
675
676 bool GameArea::buildings_selected() {
677     return buildingSelected;
678 }

```

jun 27, 17 14:46UnitsEnum.hPage 1/1

```
1 #ifndef Z_TPGRUPAL_UNITSENUM_H
2 #define Z_TPGRUPAL_UNITSENUM_H
3
4 enum class UnitsEnum {
5     /* robots */
6     /* GENERIC_ROBOT is used to draw walking and standing robots, whose
7      * animations don't differ */
8     GRUNT, PSYCHO, TOUGH, PYRO, SNIPER, LASER, GENERIC_ROBOT,
9     /* vehicles */
10    JEEP, MEDIUM_TANK, LIGHT_TANK, HEAVY_TANK, MML
11 };
12
13 #endif //Z_TPGRUPAL_UNITSENUM_H
```

jun 27, 17 14:46TeamEnum.hPage 1/1

```
1 #ifndef Z_TPGRUPAL_TEAMENUM_H
2 #define Z_TPGRUPAL_TEAMENUM_H
3
4 enum class TeamEnum {
5     NEUTRAL, BLUE, GREEN, RED, YELLOW,
6 };
7
8 #endif //Z_TPGRUPAL_TEAMENUM_H
```

jun 27, 17 14:46

**RotationsEnum.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_ROTATIONSENUM_H
2  #define Z_TPGRUPAL_ROTATIONSENUM_H
3
4  enum class RotationsEnum {
5      /* the numbers indicate the angle in which the unit is facing.
6       * 0:   looking to the right
7       * 45:  looking to upper right corner
8       * 90:  looking up
9       * 135: looking to upper left corner
10      * 180: looking to the left
11      * 225: looking to bottom left corner
12      * 270: looking down
13      * 315: looking to bottom right corner
14      * */
15
16      r000, r045, r090, r135, r180, r225, r270, r315
17  };
18
19  #endif //Z_TPGRUPAL_ROTATIONSENUM_H

```

jun 27, 17 14:46

**NatureEnum.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_NATUREENUM_H
2  #define Z_TPGRUPAL_NATUREENUM_H
3
4  enum class NatureEnum {
5      ROCK
6  };
7
8  #endif //Z_TPGRUPAL_NATUREENUM_H

```

jun 27, 17 14:46

**BuildingsEnum.h**

Page 1/1

```
1 #ifndef Z_TPGRUPAL_BUILDINGSENUM_H
2 #define Z_TPGRUPAL_BUILDINGSENUM_H
3
4 enum class BuildingsEnum {
5     FORT,
6     FORT_DESTROYED,
7     VEHICLE_FABRIC,
8     VEHICLE_FABRIC_DESTROYED,
9     ROBOT_FABRIC,
10    ROBOT_FABRI_DESTROYED
11 };
12
13 #endif //Z_TPGRUPAL_BUILDINGSENUM_H
```

jun 27, 17 14:46

**ActionsEnum.h**

Page 1/1

```
1 #ifndef Z_TPGRUPAL_ACTIONSENUM_H
2 #define Z_TPGRUPAL_ACTIONSENUM_H
3
4 enum class ActionsEnum {
5     /* REFACTOR NOTE: WALK -> MOVE*/
6     FIRE, MOVE, STAND
7 };
8
9 #endif //Z_TPGRUPAL_ACTIONSENUM_H
```



jun 27, 17 14:46

## Counter.h

Page 1/1

```
1  #ifndef Z_TPGRUPAL_COUNTER_H
2  #define Z_TPGRUPAL_COUNTER_H
3
4
5  class Counter {
6      unsigned short counter;
7      unsigned short maxSize;
8  public:
9      void initialize(unsigned long maxSize);
10
11      void updateCounter();
12
13      unsigned short getCounter() const;
14 };
15
16
17 #endif //Z_TPGRUPAL_COUNTER_H
```

jun 27, 17 14:46

## Counter.cpp

Page 1/1

```
1  #include "Counter.h"
2
3  void Counter::initialize(unsigned long maxSize) {
4      this->maxSize = maxSize;
5  }
6
7  void Counter::updateCounter() {
8      counter = maxSize - 1 ? (counter = 0) : (counter++);
9  }
10
11  unsigned short Counter::getCounter() const {
12      return counter;
13  }
```

jun 27, 17 14:46

**Winner.h**

Page 1/1

```
1  #ifndef Z_TPGRUPAL_WINNER_H
2  #define Z_TPGRUPAL_WINNER_H
3
4
5  #include "Command.h"
6  #include "../windows/GameWindow.h"
7  #include "../MapMonitor.h"
8
9  class Winner : public Command {
10      MapMonitor& map;
11      GameWindow& window;
12
13  public:
14      Winner(MapMonitor& map, GameWindow& window);
15      void execute(const std::vector<std::string> &args);
16
17  };
18
19
20 #endif //Z_TPGRUPAL_WINNER_H
```

jun 27, 17 14:46

**Winner.cpp**

Page 1/1

```
1  #include "Winner.h"
2
3  Winner::Winner(MapMonitor& map, GameWindow &window) :
4      map(map), window(window) {
5  }
6
7  void Winner::execute(const std::vector<std::string> &args) {
8      map.finish_winner();
9      window.hide();
10 }
```

jun 27, 17 14:46

## UpdateUnit.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_UPDATE_H
2  #define Z_TPGRUPAL_UPDATE_H
3
4
5  #include "Command.h"
6  #include "../UnitsMonitor.h"
7  #include "../MapMonitor.h"
8  #include "../ServerMessenger.h"
9
10 class UpdateUnit : public Command {
11     UnitsMonitor& units;
12 public:
13     explicit UpdateUnit(UnitsMonitor &units);
14
15     void execute(const std::vector<std::string> &args);
16 };
17
18
19 #endif //Z_TPGRUPAL_UPDATE_H

```

jun 27, 17 14:46

## UpdateUnit.cpp

Page 1/1

```

1  #include <iostream>
2  #include "UpdateUnit.h"
3
4  #define ID 0
5  #define STATE 1
6  #define X 2
7  #define Y 3
8  #define HP 4
9
10 const std::map<std::string, ActionsEnum> states = {
11     {std::string("atk"), ActionsEnum::FIRE},
12     {std::string("mv"), ActionsEnum::MOVE},
13     {std::string("std"), ActionsEnum::STAND}
14 };
15
16 UpdateUnit::UpdateUnit(UnitsMonitor &units) :
17     units(units)
18 {
19 }
20
21 void UpdateUnit::execute(const std::vector<std::string> &args) {
22     int id = std::stoi(args[ID]);
23     int x = std::stoi(args[X]);
24     int y = std::stoi(args[Y]);
25     ActionsEnum state = states.find(args[STATE])->second;
26     unsigned int hp = std::stoul(args[HP]);
27     units.update_position(id, state, x, y);
28     units.update_health(id, hp);
29 }

```

jun 27, 17 14:46

## UpdateTerritory.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_UPDATETERRITORY_H
2  #define Z_TPGRUPAL_UPDATETERRITORY_H
3
4
5  #include "Command.h"
6  #include "../MapMonitor.h"
7
8  class UpdateTerritory : public Command{
9      MapMonitor& mapMonitor;
10 public:
11     UpdateTerritory(MapMonitor& mapMonitor1);
12     void execute(const std::vector<std::string> &args);
13
14 };
15
16
17 #endif //Z_TPGRUPAL_UPDATETERRITORY_H

```

jun 27, 17 14:46

## UpdateTerritory.cpp

Page 1/1

```

1  #include <map>
2  #include "UpdateTerritory.h"
3  #include "../enums/TeamEnum.h"
4
5  #define ID 0
6  #define TEAM 1
7  #define X 2
8  #define Y 3
9
10 const std::map<std::string, TeamEnum> teams = {
11     {std::string("blue"), TeamEnum::BLUE},
12     {std::string("green"), TeamEnum::GREEN},
13     {std::string("neutral"), TeamEnum::NEUTRAL},
14     {std::string("red"), TeamEnum::RED},
15     {std::string("yellow"), TeamEnum::YELLOW},
16 };
17
18 UpdateTerritory::UpdateTerritory(MapMonitor &mapMonitor1)
19     : mapMonitor(mapMonitor1){
20 }
21
22 void UpdateTerritory::execute(const std::vector<std::string> &args) {
23     int id = std::stoi(args[ID]);
24     TeamEnum team = (TeamEnum) mapMonitor.get_player_id(args[TEAM]);
25     teams.find(args[TEAM])->second;
26     int x = std::stoi(args[X]);
27     int y = std::stoi(args[Y]);
28
29     mapMonitor.update_territory(id, team, x, y);
30
31     //    updateterritory-[id]-[team]-[x]-[y]
32 }

```

jun 27, 17 14:46

## UpdatePosition.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_UPDATEPOSITION_H
2  #define Z_TPGRUPAL_UPDATEPOSITION_H
3
4  #include <string>
5  #include <vector>
6  #include "Command.h"
7  #include "../UnitsMonitor.h"
8  #include "../MapMonitor.h"
9  #include "../ServerMessenger.h"
10
11 class UpdatePosition : public Command {
12     UnitsMonitor &units;
13 public:
14     explicit UpdatePosition(UnitsMonitor &units);
15
16     void execute(const std::vector<std::string> &args);
17 };
18
19 #endif //Z_TPGRUPAL_UPDATEPOSITION_H

```

jun 27, 17 14:46

## UpdatePosition.cpp

Page 1/1

```

1  #include "UpdatePosition.h"
2  #include "../MapMonitor.h"
3  #include "../ServerMessenger.h"
4
5  #define UNIT_ID 0
6  #define POS_X 1
7  #define POS_Y 2
8
9  UpdatePosition::UpdatePosition(UnitsMonitor &monitor) :
10     units(monitor)
11 {
12 }
13
14 void UpdatePosition::execute(const std::vector<std::string> &args) {
15     int id = std::stoi(args[UNIT_ID]);
16     int x = std::stoi(args[POS_X]);
17     int y = std::stoi(args[POS_Y]);
18     units.update_position(id, x, y);
19 }

```

jun 27, 17 14:46

## UpdateFactory.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_UPDATEFACTORY_H
2  #define Z_TPGRUPAL_UPDATEFACTORY_H
3
4
5  #include "Command.h"
6  #include "../BuildingsMonitor.h"
7  #include "../MapMonitor.h"
8
9  class UpdateFactory : public Command {
10      BuildingsMonitor& buildings;
11      MapMonitor& map;
12  public:
13      UpdateFactory(BuildingsMonitor& buildings, MapMonitor& map);
14      void execute(const std::vector<std::string> &args);
15  };
16
17
18
19 #endif //Z_TPGRUPAL_UPDATEFACTORY_H

```

jun 27, 17 14:46

## UpdateFactory.cpp

Page 1/1

```

1  #include "UpdateFactory.h"
2
3  #define ID 0
4  #define MINUTES 1
5  #define SECONDS 2
6  #define HP 3
7  #define TEAM 4
8
9  UpdateFactory::UpdateFactory(BuildingsMonitor &buildings, MapMonitor& map) :
10      buildings(buildings),
11      map(map)
12  {
13  }
14
15  void UpdateFactory::execute(const std::vector<std::string> &args) {
16      int id = std::stoi(args[ID]);
17      int min = std::stoi(args[MINUTES]);
18      int sec = std::stoi(args[SECONDS]);
19      int hp = std::stoi(args[HP]);
20      std::string team = args[TEAM];
21
22      TeamEnum team_enum = (TeamEnum) map.get_player_id(team);
23      buildings.update_building(id, min, sec, hp, team_enum);
24  }
25

```

jun 27, 17 14:46

**StartGame.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_STARTGAME_H
2  #define Z_TPGRUPAL_STARTGAME_H
3
4
5  #include "Command.h"
6  #include "../windows/GameWindow.h"
7  #include "../ServerMessenger.h"
8  #include "../windows/LobbyWindow.h"
9
10 class StartGame : public Command {
11     ServerMessenger &m;
12     LobbyWindow& lobby;
13     GameWindow& window;
14
15 public:
16
17     StartGame(ServerMessenger& m, LobbyWindow& lobby, GameWindow& window);
18     void execute(const std::vector<std::string> &args);
19
20 };
21
22
23 #endif //Z_TPGRUPAL_STARTGAME_H

```

jun 27, 17 14:46

**StartGame.cpp**

Page 1/1

```

1  #include "StartGame.h"
2
3  StartGame::StartGame(ServerMessenger &m,
4                      LobbyWindow& lobby,
5                      GameWindow &window) :
6      m(m),
7      window(window),
8      lobby(lobby) {
9  }
10
11 void StartGame::execute(const std::vector<std::string> &args) {
12     lobby.start_game();
13 }
14

```

jun 27, 17 14:46

## RemoveUnit.h

Page 1/1

```
1 #ifndef Z_TPGRUPAL_REMOVEUNIT_H
2 #define Z_TPGRUPAL_REMOVEUNIT_H
3
4 #include <string>
5 #include <vector>
6 #include "Command.h"
7 #include "../UnitsMonitor.h"
8
9 class RemoveUnit : public Command {
10     UnitsMonitor &monitor;
11 public:
12     explicit RemoveUnit(UnitsMonitor &monitor);
13
14     void execute(const std::vector<std::string> &args);
15 };
16
17
18 #endif //Z_TPGRUPAL_REMOVEUNIT_H
```

jun 27, 17 14:46

## RemoveUnit.cpp

Page 1/1

```
1 #include "RemoveUnit.h"
2
3 #define UNIT_ID 0
4
5 RemoveUnit::RemoveUnit(UnitsMonitor &monitor) : monitor(monitor) {
6 }
7
8 void RemoveUnit::execute(const std::vector<std::string> &args) {
9     monitor.removeUnit(std::stoi(args[UNIT_ID]));
10 }
```



jun 27, 17 14:46

## PlayerNames.h

Page 1/1

```
1 #ifndef Z_TPGRUPAL_PLAYERNAMES_H
2 #define Z_TPGRUPAL_PLAYERNAMES_H
3
4
5 #include "Command.h"
6 #include "../windows/LobbyWindow.h"
7
8 class PlayerNames : public Command {
9     LobbyWindow& lobby;
10 public:
11     PlayerNames(LobbyWindow& lobby);
12     void execute(const std::vector<std::string> &args);
13
14 };
15
16
17 #endif //Z_TPGRUPAL_PLAYERNAMES_H
```

jun 27, 17 14:46

## PlayerNames.cpp

Page 1/1

```
1 #include "PlayerNames.h"
2
3
4 PlayerNames::PlayerNames(LobbyWindow &lobby) : lobby(lobby) {
5 }
6
7 void PlayerNames::execute(const std::vector<std::string> &args) {
8     for (int i = 0; i < args.size(); ++i) {
9         lobby.update_player_name(i, args[i]);
10    }
11 }
```

jun 27, 17 14:46

**MapsInfo.h**

Page 1/1

```
1 #ifndef Z_TPGRUPAL_MAPSINFO_H
2 #define Z_TPGRUPAL_MAPSINFO_H
3
4
5 #include "Command.h"
6 #include "../windows/LobbyWindow.h"
7
8 class MapsInfo : public Command {
9     LobbyWindow& lobby;
10 public:
11     MapsInfo(LobbyWindow& lobby);
12     void execute(const std::vector<std::string> &args);
13
14 };
15
16
17 #endif //Z_TPGRUPAL_MAPSINFO_H
```

jun 27, 17 14:46

**MapsInfo.cpp**

Page 1/1

```
1 #include "MapsInfo.h"
2
3 MapsInfo::MapsInfo(LobbyWindow &lobby) : lobby(lobby) {
4 }
5
6 void MapsInfo::execute(const std::vector<std::string> &args) {
7     std::stringstream s;
8     for (const std::string& map : args) {
9         s << map << std::endl;
10    }
11    lobby.update_maps(s.str());
12 }
```

jun 27, 17 14:46

**Loser.h**

Page 1/1

```
1  #ifndef Z_TPGRUPAL_LOSER_H
2  #define Z_TPGRUPAL_LOSER_H
3
4
5  #include "Command.h"
6  #include "../windows/GameWindow.h"
7
8  class Loser : public Command {
9      MapMonitor& map;
10     GameWindow& window;
11 public:
12     Loser(MapMonitor& map, GameWindow &window);
13     void execute(const std::vector<std::string> &args);
14 };
15
16
17 #endif //Z_TPGRUPAL_LOSER_H
```

jun 27, 17 14:46

**Loser.cpp**

Page 1/1

```
1  #include "Loser.h"
2
3
4  Loser::Loser(MapMonitor& map, GameWindow &window) : map(map), window(window) {
5  }
6
7  void Loser::execute(const std::vector<std::string> &args) {
8      map.finish_loser();
9      window.hide();
10 }
```

jun 27, 17 14:46

**LobbyInfo.h**

Page 1/1

```
1  #ifndef Z_TPGRUPAL_LOBBYINFO_H
2  #define Z_TPGRUPAL_LOBBYINFO_H
3
4
5  #include "Command.h"
6  #include "../windows/MenuWindow.h"
7
8  class LobbyInfo : public Command {
9      MenuWindow& menu;
10 public:
11     LobbyInfo(MenuWindow& menu);
12     void execute(const std::vector<std::string> &args);
13
14 };
15
16
17 #endif //Z_TPGRUPAL_LOBBYINFO_H
```

jun 27, 17 14:46

**LobbyInfo.cpp**

Page 1/1

```
1  #include "LobbyInfo.h"
2
3  LobbyInfo::LobbyInfo(MenuWindow &menu) : menu(menu) {
4  }
5
6  void LobbyInfo::execute(const std::vector<std::string> &args) {
7      menu.update_lobbies(args);
8  }
```

jun 27, 17 14:46

## LoadMap.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_LOADMAP_H
2  #define Z_TPGRUPAL_LOADMAP_H
3
4
5  #include "../MapMonitor.h"
6  #include "Command.h"
7  #include "../BuildingsMonitor.h"
8  #include "../windows/GameWindow.h"
9  #include <string>
10 #include <vector>
11
12 class LoadMap : public Command {
13     MapMonitor &mapMonitor;
14     BuildingsMonitor &buildingsMonitor;
15     GameWindow &window;
16
17 public:
18     LoadMap(MapMonitor &monitor, BuildingsMonitor &buildingsMonitor,
19             GameWindow &window);
20
21     void execute(const std::vector<std::string> &args);
22 };
23
24
25 #endif //Z_TPGRUPAL_LOADMAP_H

```

jun 27, 17 14:46

## LoadMap.cpp

Page 1/1

```

1  #include <pugixml.hpp>
2  #include <iostream>
3  #include "LoadMap.h"
4
5
6  #define ROBOT_FACTORY "res/assets/buildings/robot/base_jungle.png"
7  #define VEHICLE_FACTORY "res/assets/buildings/vehicle/base_jungle.png"
8
9  const std::map<std::string, BuildingsEnum> buildings{
10     {std::string("VehicleFactory"), BuildingsEnum::VEHICLE_FABRIC},
11     {std::string("UnitFactory"), BuildingsEnum::ROBOT_FABRIC},
12     {std::string("Fort"), BuildingsEnum::FORT}
13 };
14
15 LoadMap::LoadMap(MapMonitor &mapMonitor, BuildingsMonitor &buildings,
16                 GameWindow &window) :
17     mapMonitor(mapMonitor), buildingsMonitor(buildings), window(window) {}
18
19
20 void LoadMap::execute(const std::vector<std::string> &args) {
21     /* initialize map so then can be completed with read data */
22     mapMonitor.initializeMap(100, 100);
23
24     pugi::xml_document doc;
25     /* the only arg we receive is the map,
26      * which is the whole xml saved in a string */
27     pugi::xml_parse_result result = doc.load_string(args[0].c_str());
28     if (!result) {
29         /* FATAL ERROR LOADING MAP */
30         return;
31     }
32
33     std::vector<std::vector<std::string>> map;
34     pugi::xml_node root = doc.child("Map");
35     pugi::xml_node terrain_node = root.child("Terrain");
36     for (auto node_row : terrain_node.children()) {
37         unsigned int coord_x = 0;
38         for (auto cell : node_row.children()) {
39             if (map.size() <= coord_x) {
40                 map.push_back(std::vector<std::string>());
41             }
42             std::string terrain = cell.attribute("terrain").value();
43
44             map.at(coord_x++).push_back(terrain);
45         }
46     }
47
48     unsigned long size = map.size();
49     mapMonitor.initializeMap(size, size);
50     for (int i = 0; i < size; i++) {
51         for (int j = 0; j < size; j++) {
52             mapMonitor.setCell(i, j, map[i][j]);
53         }
54     }
55     window.setMapData();
56 }

```

jun 27, 17 14:46

## JoinLobby.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_JOINLOBBY_H
2  #define Z_TPGRUPAL_JOINLOBBY_H
3
4
5  #include "Command.h"
6  #include "../windows/MenuWindow.h"
7  #include "../windows/LobbyWindow.h"
8
9  class JoinLobby : public Command {
10      MenuWindow& menu;
11      LobbyWindow& lobby;
12      ServerMessenger& messenger;
13  public:
14      JoinLobby(MenuWindow& menu, LobbyWindow& lobby, ServerMessenger& messenger);
15      void execute(const std::vector<std::string> &args);
16  };
17
18
19 #endif //Z_TPGRUPAL_JOINLOBBY_H

```

jun 27, 17 14:46

## JoinLobby.cpp

Page 1/1

```

1  #include "JoinLobby.h"
2
3  #define STATUS 0
4
5  #define OK "ok"
6  #define ERROR "error"
7  JoinLobby::JoinLobby(MenuWindow &menu, LobbyWindow &lobby,
8                      ServerMessenger& messenger) :
9      menu(menu),
10     lobby(lobby),
11     messenger(messenger)
12 {
13 }
14
15 void JoinLobby::execute(const std::vector<std::string> &args) {
16     if (args[STATUS] == OK) {
17         menu.join_lobby();
18
19         // Fetch available maps
20         messenger.send("mapsinfo");
21     }
22 }

```

jun 27, 17 14:46

**FactoryStats.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_FACTORYSTATS_H
2  #define Z_TPGRUPAL_FACTORYSTATS_H
3
4
5  #include "Command.h"
6  #include "../BuildingsMonitor.h"
7  #include "../windows/GameWindow.h"
8
9  class FactoryStats : public Command {
10     GameWindow& window;
11 public:
12     FactoryStats(GameWindow& window);
13     void execute(const std::vector<std::string> &args);
14
15 };
16
17
18 #endif //Z_TPGRUPAL_FACTORYSTATS_H

```

jun 27, 17 14:46

**FactoryStats.cpp**

Page 1/1

```

1  #include "FactoryStats.h"
2  #include "../windows/GameWindow.h"
3
4  FactoryStats::FactoryStats(GameWindow& window) : window(window) {
5  }
6
7  #define TYPE 0
8  #define FIRE_RATE 1
9  #define MINUTES 2
10 #define SECONDS 3
11 #define HP 4
12
13 void FactoryStats::execute(const std::vector<std::string> &args) {
14     std::string type = args[TYPE];
15     int fire_rate = std::stoi(args[FIRE_RATE]);
16     int hp = std::stoi(args[HP]);
17     int minutes = std::stoi(args[MINUTES]);
18     int seconds = std::stoi(args[SECONDS]);
19     window.update_factory_panel(type, fire_rate, hp);
20     window.update_factory_timer(minutes, seconds);
21 }

```

jun 27, 17 14:46

**FactoryNextUnit.h**

Page 1/1

```
1  #ifndef Z_TPGRUPAL_FACTORYNEXTUNIT_H
2  #define Z_TPGRUPAL_FACTORYNEXTUNIT_H
3
4
5  #include "Command.h"
6  #include "../windows/GameWindow.h"
7
8  class FactoryNextUnit : public Command {
9      GameWindow &window;
10 public:
11     explicit FactoryNextUnit(GameWindow &window);
12
13     void execute(const std::vector<std::string> &args);
14 };
15
16
17 #endif //Z_TPGRUPAL_FACTORYNEXTUNIT_H
```

jun 27, 17 14:46

**FactoryNextUnit.cpp**

Page 1/1

```
1  #include "FactoryNextUnit.h"
2
3  #define UNIT_NAME 0
4
5  FactoryNextUnit::FactoryNextUnit(GameWindow &window) : window(window) {
6  }
7
8  void FactoryNextUnit::execute(const std::vector<std::string> &args) {
9      std::string path = "res/portraits/" + args[UNIT_NAME] + ".png";
10     window.factory_change_unit(path);
11 }
12
13
```



jun 27, 17 14:46

**Command.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_COMMAND_H
2  #define Z_TPGRUPAL_COMMAND_H
3
4  #include <vector>
5  #include <string>
6
7  class Command {
8  public:
9      virtual void execute(const std::vector<std::string> &args) = 0;
10 };
11
12 #endif //Z_TPGRUPAL_COMMAND_H

```

jun 27, 17 14:46

**AddUnit.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_ADDUNIT_H
2  #define Z_TPGRUPAL_ADDUNIT_H
3
4  #include <vector>
5  #include <string>
6  #include "Command.h"
7  #include "../UnitsMonitor.h"
8  #include "../MapMonitor.h"
9
10 class AddUnit : public Command {
11     UnitsMonitor &unitsMonitor;
12     MapMonitor& map;
13 public:
14     explicit AddUnit(UnitsMonitor &unitsMonitor,
15                     MapMonitor& map);
16
17     void execute(const std::vector<std::string> &args);
18 };
19
20
21 #endif //Z_TPGRUPAL_ADDUNIT_H

```

jun 27, 17 14:46

## AddUnit.cpp

Page 1/1

```

1  #include <iostream>
2  #include <vector>
3  #include "AddUnit.h"
4
5  #define UNIT_ID 0
6  #define X 1
7  #define Y 2
8  #define UNIT_NAME 3
9  #define TEAM 4
10 #define HP 5
11
12 const std::map<std::string, UnitsEnum> units = {
13     {std::string("grunt"), UnitsEnum::GRUNT},
14     {std::string("psycho"), UnitsEnum::PSYCHO},
15     {std::string("tough"), UnitsEnum::TOUGH},
16     {std::string("pyro"), UnitsEnum::PYRO},
17     {std::string("sniper"), UnitsEnum::SNIPER},
18     {std::string("laser"), UnitsEnum::LASER},
19     {std::string("generic_robot"), UnitsEnum::GENERIC_ROBOT},
20     {std::string("jeep"), UnitsEnum::JEEP},
21     {std::string("medium_tank"), UnitsEnum::MEDIUM_TANK},
22     {std::string("light_tank"), UnitsEnum::LIGHT_TANK},
23     {std::string("heavy_tank"), UnitsEnum::HEAVY_TANK},
24     {std::string("mml"), UnitsEnum::MML}
25 };
26
27 void AddUnit::execute(const std::vector<std::string> &args) {
28     int x = std::stoi(args[X]);
29     int y = std::stoi(args[Y]);
30     int id = std::stoi(args[UNIT_ID]);
31
32     std::string name = args[UNIT_NAME];
33     auto type = units.find(name);
34     if (type == units.end()) {
35         /* Error adding unit: received type */
36         return;
37     }
38
39     std::string owner = args[TEAM];
40     int team_id = map.get_player_id(owner);
41     unsigned int hp = std::stoul(args[HP]);
42     Unit unit(id, {x, y}, type->second, (TeamEnum) team_id, hp);
43     unit.update_owner(owner);
44     unit.update_unit_name(name);
45
46     unitsMonitor.addUnit(unit);
47 }
48
49 AddUnit::AddUnit(UnitsMonitor &unitsMonitor,
50                 MapMonitor& map)
51     : unitsMonitor(unitsMonitor),
52       map(map)
53 {
54 }

```

jun 27, 17 14:46

## AddNature.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_ADDNATURE_H
2  #define Z_TPGRUPAL_ADDNATURE_H
3
4
5  #include "Command.h"
6  #include "../MapMonitor.h"
7
8  class AddNature : public Command {
9      MapMonitor &mapMonitor;
10 public:
11     explicit AddNature(MapMonitor &mapMonitor);
12
13     void execute(const std::vector<std::string> &args);
14 };
15
16
17 #endif //Z_TPGRUPAL_ADDNATURE_H

```

jun 27, 17 14:46

## AddNature.cpp

Page 1/1

```

1  #include <map>
2  #include "AddNature.h"
3
4  #define ID 0
5  #define X 1
6  #define Y 2
7  #define TYPE 3
8  #define TEAM 4
9
10 const std::map<std::string, NatureEnum > natureMap = {
11     {std::string("Rock"), NatureEnum ::ROCK}
12 };
13
14 AddNature::AddNature(MapMonitor &mapMonitor) : mapMonitor(mapMonitor){
15 }
16
17 void AddNature::execute(const std::vector<std::string> &args) {
18     int id = std::stoi(args[ID]);
19     int x = std::stoi(args[X]);
20     int y = std::stoi(args[Y]);
21     NatureEnum natureType = natureMap.find(args[TYPE])→second;
22
23     Nature nature(natureType, {x, y}, id);
24
25     mapMonitor.addNature(nature);
26     // Building b(BuildingsEnum::ROBOT_FABRIC, x, y, id, team);
27     // buildings.addBuilding(b);
28 }
29

```

jun 27, 17 14:46

## AddBuilding.h

Page 1/1

```

1  #ifndef Z_TPGRUPAL_ADDBUILDING_H
2  #define Z_TPGRUPAL_ADDBUILDING_H
3
4
5  #include "../BuildingsMonitor.h"
6  #include "Command.h"
7  #include "../MapMonitor.h"
8
9  class AddBuilding : public Command {
10     BuildingsMonitor &buildings;
11     MapMonitor& map;
12 public:
13     explicit AddBuilding(BuildingsMonitor &buildings,
14                         MapMonitor& map);
15
16     void execute(const std::vector<std::string> &args);
17 };
18
19
20 #endif //Z_TPGRUPAL_ADDBUILDING_H

```

jun 27, 17 14:46

**AddBuilding.cpp**

Page 1/1

```

1  #include "AddBuilding.h"
2
3  #define TILESIZE 16    //tile width in pixels.
4  #define ID 0
5  #define X 1
6  #define Y 2
7  #define TYPE 3
8  #define TEAM 4
9  #define HP 5
10
11 const std::map<std::string, BuildingsEnum> buildingsMap = {
12     {std::string("Fort"), BuildingsEnum::FORT},
13     /* Since we dont yet distinguish between robot and vehicle
14      * factories, all factories will be drawn as robot factories. */
15     {std::string("Factory"), BuildingsEnum::ROBOT_FABRIC}
16 };
17
18 AddBuilding::AddBuilding(BuildingsMonitor &buildings,
19                          MapMonitor& map)
20     : buildings(buildings),
21       map(map)
22 {
23 }
24
25 void AddBuilding::execute(const std::vector<std::string> &args) {
26     int id = std::stoi(args[ID]);
27     int x = std::stoi(args[X]);
28     int y = std::stoi(args[Y]);
29
30     BuildingsEnum building_type = buildingsMap.find(args[TYPE])->second;
31
32     std::string owner = args[TEAM];
33     int team_id = map.get_player_id(owner);
34
35     unsigned int hp = std::stoul(args[HP]);
36     Building b(building_type, x, y, id, (TeamEnum) team_id, owner, hp);
37     buildings.addBuilding(b);
38 }

```

jun 27, 17 14:46

**ClientThread.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_CLIENTTREAD_H
2  #define Z_TPGRUPAL_CLIENTTREAD_H
3
4  #include <map>
5  #include <string>
6  #include "BuildingsMonitor.h"
7  #include "MapMonitor.h"
8  #include "ServerMessenger.h"
9  #include "commands/Command.h"
10 #include "UnitsMonitor.h"
11 #include "windows/GameWindow.h"
12 #include "windows/LobbyWindow.h"
13 #include "GameBuilder.h"
14 #include <Thread.h>
15
16 class ClientThread : public Thread {
17     UnitsMonitor &unitsMonitor;
18     BuildingsMonitor &buildingsMonitor;
19     MapMonitor &mapMonitor;
20     ServerMessenger &messenger;
21     GameWindow &window;
22     MenuWindow& menu;
23     LobbyWindow& lobby;
24     bool finished = false; // Flag for finishing execution
25     std::map<std::string, Command *> commands;
26 public:
27     ClientThread(UnitsMonitor &unitsMonitor,
28                 BuildingsMonitor &buildingsMonitor,
29                 MapMonitor &mapMonitor, ServerMessenger &messenger,
30                 GameBuilder& builder);
31
32     virtual void run();
33
34     ~ClientThread();
35
36     /* Finish the thread's execution */
37     void finish();
38 private:
39     /* Loads commands */
40     void initCommands();
41
42     /** Main thread loop, receives commands from the server, parses them and
43      * executes them */
44     void loop();
45
46     /* Parses a command string and executes it */
47     void parse(std::string &s);
48
49 };
50
51
52
53 #endif //Z_TPGRUPAL_CLIENTTREAD_H

```

jun 27, 17 14:46

## ClientThread.cpp

Page 1/2

```

1  #include <iostream>
2  #include <vector>
3  #include <pugixml.hpp>
4  #include <split.h>
5  #include <utility>
6  #include "ClientThread.h"
7  #include "commands/AddUnit.h"
8  #include "commands/RemoveUnit.h"
9  #include "commands/UpdatePosition.h"
10 #include "commands/LoadMap.h"
11 #include "commands/UpdateUnit.h"
12 #include "commands/FactoryNextUnit.h"
13 #include "commands/AddBuilding.h"
14 #include "commands/AddNature.h"
15 #include "commands/StartGame.h"
16 #include "commands/PlayerNames.h"
17 #include "commands/UpdateFactory.h"
18 #include "commands/FactoryStats.h"
19 #include "commands/LobbyInfo.h"
20 #include "commands/JoinLobby.h"
21 #include "commands/Winner.h"
22 #include "commands/Loser.h"
23 #include "commands/MapsInfo.h"
24 #include "commands/UpdateTerritory.h"
25
26 void ClientThread::run() {
27     initCommands();
28     loop();
29 }
30
31 ClientThread::ClientThread(UnitsMonitor &unitsMonitor,
32                             BuildingsMonitor &buildingsMonitor,
33                             MapMonitor &mapMonitor,
34                             ServerMessenger &messenger,
35                             GameBuilder& builder) :
36     unitsMonitor(unitsMonitor),
37     buildingsMonitor(buildingsMonitor),
38     mapMonitor(mapMonitor),
39     messenger(messenger),
40     lobby(*builder.get_lobby_window()),
41     window(*builder.get_window()),
42     menu(*builder.get_menu_window())
43 {
44 }
45
46 void ClientThread::loop() {
47     try {
48         while (!finished) {
49             std::string msg = messenger.receive();
50             std::vector<std::string> commands = utils::split(msg, '|');
51             for (std::string &cmd : commands) {
52                 parse(cmd);
53             }
54         }
55     } catch (SocketError &e) {
56         return;
57     }
58 }
59
60 void ClientThread::parse(std::string &s) {
61     std::vector<std::string> params = utils::split(s, '-');
62     int cmd = 0;
63     auto result = commands.find(params[cmd]);
64     if (result == commands.end()) {
65         /* Invalid command */
66         return;

```

jun 27, 17 14:46

## ClientThread.cpp

Page 2/2

```

67     }
68     std::vector<std::string> args(++params.begin(), params.end());
69     result->second->execute(args);
70 }
71
72 void ClientThread::finish() {
73     finished = true;
74     messenger.kill();
75 }
76
77 void ClientThread::initCommands() {
78     commands["loadmap"] = new LoadMap(mapMonitor, buildingsMonitor, window);
79     commands["addunit"] = new AddUnit(unitsMonitor, mapMonitor);
80     commands["removeunit"] = new RemoveUnit(unitsMonitor);
81     commands["move"] = new UpdatePosition(unitsMonitor);
82     commands["updateunit"] = new UpdateUnit(unitsMonitor);
83     commands["nextunit"] = new FactoryNextUnit(window);
84     commands["addbuilding"] = new AddBuilding(buildingsMonitor, mapMonitor);
85     commands["addnature"] = new AddNature(mapMonitor);
86     commands["startgame"] = new StartGame(messenger, lobby, window);
87     commands["names"] = new PlayerNames(lobby);
88     commands["updatefactory"] = new UpdateFactory(buildingsMonitor, mapMonitor);
89     commands["factorystats"] = new FactoryStats(window);
90     commands["lobbyinfo"] = new LobbyInfo(menu);
91     commands["joinlobby"] = new JoinLobby(menu, lobby, messenger);
92     commands["winner"] = new Winner(mapMonitor, window);
93     commands["loseryousuck"] = new Loser(mapMonitor, window);
94     commands["mapsinfo"] = new MapsInfo(lobby);
95     commands["updateterritory"] = new UpdateTerritory(mapMonitor);
96 }
97
98 ClientThread::~ClientThread() {
99     for (std::pair<std::string, Command *> c : commands) {
100         delete c.second;
101     }
102 }

```

jun 27, 17 14:46

**Cell.h**

Page 1/1

```
1 #ifndef Z_TPGRUPAL_CELL_H
2 #define Z_TPGRUPAL_CELL_H
3
4 #include <string>
5
6 class Cell {
7 private:
8     std::string terrainType;
9
10 public:
11     Cell();
12
13     explicit Cell(std::string &terrainType);
14
15     void assignTerrainType(std::string terrainType);
16
17     std::string getTerrainType();
18 };
19
20
21 #endif //Z_TPGRUPAL_CELL_H
```

jun 27, 17 14:46

**Cell.cpp**

Page 1/1

```
1 #include "Cell.h"
2
3 Cell::Cell() : terrainType("") {
4 }
5
6 Cell::Cell(std::string &terrainType) : terrainType(terrainType) {
7 }
8
9 void Cell::assignTerrainType(std::string terrainType) {
10     this->terrainType = terrainType;
11 }
12
13 std::string Cell::getTerrainType() {
14     return terrainType;
15 }
```

jun 27, 17 14:46	Camera.h	Page 1/1
1	<b>#ifndef</b> Z_TPGRUPAL_CAMERA_H	
2	<b>#define</b> Z_TPGRUPAL_CAMERA_H	
3		
4		
5	<b>#include</b> <utility>	
6		
7	class Camera {	
8	private:	
9	<i>/* save tile size in pixels so calculations on max and min coord.</i>	
10	<i>  * can be done */</i>	
11	unsigned int tileSize;	
12		
13	unsigned int numberOfTilesToShow;	
14		
15	<i>/* position in pixels */</i>	
16	std::pair<unsigned int, unsigned int> position;	
17		
18	unsigned int minXCoordinate, minYCoordinate, maxXCoordinate, maxYCoordinate;	
19		
20	unsigned int mapWidth, mapHeight;	
21		
22	public:	
23	<i>/* camera will be initialized in (minx, miny) position */</i>	
24	Camera(unsigned int tileSize, unsigned int mapWidth, unsigned int mapHeight,	
25	unsigned int numberOfTilesToShow);	
26		
27	std::pair<unsigned int, unsigned int> getPosition();	
28		
29	void setMapWidth(unsigned int width);	
30		
31	void setMapHeight(unsigned int height);	
32		
33	void moveUp();	
34		
35	void moveDown();	
36		
37	void moveLeft();	
38		
39	void moveRight();	
40		
41	std::pair<unsigned int, unsigned int> cameraOffset();	
42		
43	unsigned int idealMapToCameraXCoordinate(unsigned int globalXPosition);	
44		
45	unsigned int idealMapToCameraYCoordinate(unsigned int globalYPosition);	
46		
47	unsigned int cameraToMapXCoordinate(unsigned int coordinate);	
48		
49	unsigned int cameraToMapYCoordinate(unsigned int coordinate);	
50	};	
51		
52	<b>#endif</b> //Z_TPGRUPAL_CAMERA_H	
53		

jun 27, 17 14:46	Camera.cpp	Page 1/2
1	<b>#include</b> "Camera.h"	
2	<b>#include</b> <iostream>	
3		
4	Camera::Camera(unsigned int tileSize, unsigned int mapWidth,	
5	unsigned int mapHeight,	
6	unsigned int numberOfTilesToShow) :	
7	tileSize(tileSize),	
8	numberOfTilesToShow(numberOfTilesToShow),	
9	minXCoordinate(numberOfTilesToShow * tileSize / 2),	
10	minYCoordinate(numberOfTilesToShow * tileSize / 2),	
11	maxXCoordinate(	
12	mapWidth * tileSize - numberOfTilesToShow * tileSize / 2),	
13	maxYCoordinate(	
14	mapHeight * tileSize - numberOfTilesToShow * tileSize / 2),	
15	position(numberOfTilesToShow * tileSize / 2,	
16	numberOfTilesToShow * tileSize / 2) {	
17	}	
18		
19	/**	
20	*	
21	* @return camera position is given in pixels.	
22	*/	
23	std::pair<unsigned int, unsigned int> Camera::getPosition() {	
24	return position;	
25	}	
26		
27	void Camera::setMapWidth(unsigned int width) {	
28	mapWidth = width;	
29	maxXCoordinate = mapWidth * tileSize - numberOfTilesToShow * tileSize / 2;	
30	}	
31		
32	void Camera::setMapHeight(unsigned int height) {	
33	mapHeight = height;	
34	maxYCoordinate = mapHeight * tileSize - numberOfTilesToShow * tileSize / 2;	
35	}	
36		
37	void Camera::moveUp() {	
38	if (position.second != minYCoordinate) {	
39	position.second -= tileSize;	
40	}	
41	}	
42		
43	void Camera::moveDown() {	
44	if (position.second != maxYCoordinate) {	
45	position.second += tileSize;	
46	}	
47	}	
48		
49	void Camera::moveRight() {	
50	if (position.first != maxXCoordinate) {	
51	position.first += tileSize;	
52	}	
53	}	
54		
55	void Camera::moveLeft() {	
56	if (position.first != minXCoordinate) {	
57	position.first -= tileSize;	
58	}	
59	}	
60		
61	std::pair<unsigned int, unsigned int> Camera::cameraOffset() {	
62	return std::pair<unsigned int, unsigned int>	
63	(position.first - numberOfTilesToShow * tileSize / 2,	
64	position.second - numberOfTilesToShow * tileSize / 2);	
65	}	
66		

jun 27, 17 14:46

**Camera.cpp**

Page 2/2

```

67 unsigned int Camera::idealMapToCameraXCoordinate(unsigned int globalXPosition) {
68     return globalXPosition - cameraOffset().first;
69 }
70
71 unsigned int Camera::idealMapToCameraYCoordinate(unsigned int globalYPosition) {
72     return globalYPosition - cameraOffset().second;
73 }
74
75 unsigned int Camera::cameraToMapXCoordinate(unsigned int coordinate) {
76     return coordinate + cameraOffset().first;
77 }
78
79 unsigned int Camera::cameraToMapYCoordinate(unsigned int coordinate) {
80     return coordinate + cameraOffset().second;
81 }

```

jun 27, 17 14:46

**BuildingsMonitor.h**

Page 1/1

```

1  #ifndef Z_TPGRUPAL_BUILDINGSMONITOR_H
2  #define Z_TPGRUPAL_BUILDINGSMONITOR_H
3
4
5  #include <mutex>
6  #include "Building.h"
7  #include <vector>
8
9  class BuildingsMonitor {
10 private:
11     std::vector<Building> buildings;
12
13     std::mutex m;
14 public:
15
16     void markAsSelectedInRange(bool &buildingSelected,
17                               gdouble xStartCoordinate,
18                               gdouble yStartCoordinate,
19                               gdouble xFinishCoordinate,
20                               gdouble yFinishCoordinate);
21
22     void addBuilding(Building &b);
23
24     std::vector<Building>
25     getBuildingsToDraw(unsigned int minX, unsigned int maxX, unsigned int minY,
26                       unsigned int maxY);
27
28     std::vector<Building> get_selected();
29     void wipe_selected();
30     Building get_building(int id);
31
32     void update_building(int id, int minutes, int seconds, int hp,
33                         TeamEnum team);
34     void clear();
35 };
36
37
38 #endif //Z_TPGRUPAL_BUILDINGSMONITOR_H

```



jun 27, 17 14:46

## BuildingsMonitor.cpp

Page 1/2

```

1  #include "BuildingsMonitor.h"
2  #include <Lock.h>
3
4
5  void BuildingsMonitor::markAsSelectedInRange(bool &buildingSelected,
6                                              gdouble xStartCoordinate,
7                                              gdouble yStartCoordinate,
8                                              gdouble xFinishCoordinate,
9                                              gdouble yFinishCoordinate) {
10
11      Lock l(m);
12      for (Building &building : buildings) {
13          /* each player has to select its units in range */
14          building.markAsSelectedInRange(buildingSelected, xStartCoordinate,
15                                         yStartCoordinate, xFinishCoordinate,
16                                         yFinishCoordinate);
17          if (buildingSelected) {
18              break;
19          }
20      }
21
22  void BuildingsMonitor::addBuilding(Building &b) {
23      Lock l(m);
24      buildings.push_back(b);
25  }
26
27  std::vector<Building>
28  BuildingsMonitor::getBuildingsToDraw(unsigned int minX, unsigned int maxX,
29                                       unsigned int minY, unsigned int maxY) {
30
31      Lock l(m);
32      std::vector<Building> returnVector;
33
34      for (Building &building : buildings) {
35          if (building.getXCoordinate() >= minX and
36              building.getXCoordinate() <= maxX and
37              building.getYCoordinate() >= minY and
38              building.getYCoordinate() <= maxY) {
39              returnVector.emplace_back(building);
40          }
41      }
42      return returnVector;
43  }
44
45  std::vector<Building> BuildingsMonitor::get_selected() {
46      std::vector<Building> selected_buildings;
47      for (Building &building : buildings) {
48          if (building.is_selected()) {
49              selected_buildings.push_back(building);
50          }
51      }
52      return selected_buildings;
53  }
54
55  void BuildingsMonitor::wipe_selected() {
56      for (Building &b: buildings) {
57          b.unselect();
58      }
59  }
60
61  Building BuildingsMonitor::get_building(int id) {
62      Lock l(m);
63      for (Building& building: buildings) {
64          if (building.get_ID() == id) {
65              return building;
66          }
67      }

```

jun 27, 17 14:46

## BuildingsMonitor.cpp

Page 2/2

```

67      return Building();
68  }
69
70  void
71  BuildingsMonitor::update_building(int id, int minutes, int seconds, int hp,
72                                   TeamEnum team) {
73      Lock l(m);
74      for (auto b = buildings.begin(); b != buildings.end(); ++b) {
75          if (b->get_ID() == id) {
76              b->update_hp(hp);
77              b->update_time_left(minutes, seconds);
78              b->update_team(team);
79          }
80      }
81  }
82
83  void BuildingsMonitor::clear() {
84      Lock l(m);
85      buildings.clear();
86  }

```

jun 27, 17 14:46	Building.h	Page 1/2
<pre> 1  #ifndef Z_TPGRUPAL_BUILDING_H 2  #define Z_TPGRUPAL_BUILDING_H 3 4  #include &lt;gtkmm/drawingarea.h&gt; 5  #include "enums/BuildingsEnum.h" 6  #include "enums/TeamEnum.h" 7  #include &lt;utility&gt; 8  #include &lt;map&gt; 9  #include &lt;string&gt; 10 #include &lt;vector&gt; 11 12 class Building { 13 private: 14     BuildingsEnum buildingType; 15     TeamEnum team; 16     std::pair&lt;unsigned int, unsigned int&gt; position; 17 18     /* bool selected: indicates wether the unit has been selected 19      * with the mouse or not */ 20     bool selected; 21     int id; 22     std::string owner; 23     int minutes; 24     int seconds; 25     unsigned int max_hp; 26     unsigned int hp; 27 public: 28     Building(BuildingsEnum type, int x, int y, int id, TeamEnum team, 29             const std::string&amp; owner, unsigned int hp); 30 31     Building(); 32     /** 33      * This methods checks all the player's units to see if any of its units is 34      * located within the area of selection. If so, the units' attribute 35      * selected is set to TRUE. 36      */ 37     void markAsSelectedInRange(bool &amp;buildingSelected, 38                               gdouble xStartCoordinate, 39                               gdouble yStartCoordinate, 40                               gdouble xFinishCoordinate, 41                               gdouble yFinishCoordinate); 42 43     unsigned int getXCoordinate(); 44 45     unsigned int getYCoordinate(); 46 47     BuildingsEnum getBuildingType(); 48 49     TeamEnum getTeam(); 50     bool is_selected(); 51     void unselect(); 52     std::string get_owner(); 53     unsigned int get_hp(); 54 55     unsigned int get_max_hp(); 56 57     int get_ID(); 58 59     void update_time_left(int minutes, int seconds); 60 61     void update_hp(unsigned int hp); 62 63     void update_team(TeamEnum team); 64 65     std::pair&lt;int, int&gt; get_time_left(); 66 </pre>		

jun 27, 17 14:46	Building.h	Page 2/2
<pre> 67         void destroy(); 68     }; 69 70 71 #endif //Z_TPGRUPAL_BUILDING_H </pre>		

jun 27, 17 14:46

## Building.cpp

Page 1/2

```

1  #include "Building.h"
2
3  #define LENIENCY_FORT 16
4  #define LENIENCY_FACTORY 8
5
6  void Building::markAsSelectedInRange(bool &buildingSelected,
7                                     gdouble xStartCoordinate,
8                                     gdouble yStartCoordinate,
9                                     gdouble xFinishCoordinate,
10                                    gdouble yFinishCoordinate) {
11
12     int x = (int) xFinishCoordinate;
13     int y = (int) yFinishCoordinate;
14     int x_abs = abs(position.first - x);
15     int y_abs = abs(position.second - y);
16
17     unsigned int leniency;
18
19     if (buildingType == BuildingsEnum::FORT or
20         buildingType == BuildingsEnum::FORT_DESTROYED) {
21         /* case building is fort... */
22         leniency = LENIENCY_FORT;
23     } else {
24         /* case building is factory... */
25         leniency = LENIENCY_FACTORY;
26     }
27     if (x_abs ≤ leniency ^ y_abs ≤ leniency) {
28         selected = true;
29         buildingSelected = true;
30     }
31 }
32
33 Building::Building(BuildingsEnum type, int x, int y, int id, TeamEnum team,
34                   const std::string& owner, unsigned int hp) :
35     buildingType(type),
36     position(x, y),
37     id(id),
38     selected(false),
39     team(team),
40     owner(owner),
41     hp(hp),
42     max_hp(hp),
43     minutes(0),
44     seconds(0)
45 {
46 }
47
48 unsigned int Building::getXCoordinate() {
49     return position.first;
50 }
51
52 unsigned int Building::getYCoordinate() {
53     return position.second;
54 }
55
56 BuildingsEnum Building::getBuildingType() {
57     return buildingType;
58 }
59
60 TeamEnum Building::getTeam() {
61     return team;
62 }
63
64 bool Building::is_selected() {
65     return selected;
66 }

```

jun 27, 17 14:46

## Building.cpp

Page 2/2

```

67
68 void Building::unselect() {
69     selected = false;
70 }
71
72 Building::Building() {
73     id = 0;
74 }
75
76 std::string Building::get_owner() {
77     return owner;
78 }
79
80 unsigned int Building::get_hp() {
81     return hp;
82 }
83
84 unsigned int Building::get_max_hp() {
85     return max_hp;
86 }
87
88 int Building::get_ID() {
89     return id;
90 }
91
92 void Building::update_time_left(int minutes, int seconds) {
93     this->minutes = minutes;
94     this->seconds = seconds;
95 }
96
97 void Building::update_hp(unsigned int hp) {
98     this->hp = hp;
99     if (hp ≤ 0) {
100         destroy();
101     }
102 }
103
104 std::pair<int, int> Building::get_time_left() {
105     return {minutes, seconds};
106 }
107
108 void Building::destroy() {
109     if (buildingType == BuildingsEnum::ROBOT_FABRIC) {
110         buildingType = BuildingsEnum::ROBOT_FABRI_DESTROYED;
111     } else if (buildingType == BuildingsEnum::VEHICLE_FABRIC) {
112         buildingType = BuildingsEnum::VEHICLE_FABRIC_DESTROYED;
113     } else if (buildingType == BuildingsEnum::FORT) {
114         buildingType = BuildingsEnum::FORT_DESTROYED;
115     }
116 }
117
118 void Building::update_team(TeamEnum team) {
119     this->team = team;
120 }

```

jun 27, 17 14:46

**Armament.h**

Page 1/1

```
1  #ifndef Z_TPGRUPAL_ARMAMENT_H
2  #define Z_TPGRUPAL_ARMAMENT_H
3
4
5  #include <vector>
6  #include <string>
7  #include <map>
8  #include <gtkmm/drawingarea.h>
9  #include <SDL2/SDL_mixer.h>
10
11 class Armament {
12 private:
13     /* sound used when shot */
14     Mix_Chunk *sound;
15
16     /* animation is stored as a series of images needed
17      * to draw the animation. */
18     std::vector<Glib::RefPtr<Gdk::Pixbuf>> animation;
19 };
20
21
22 #endif //Z_TPGRUPAL_ARMAMENT_H
```

jun 27, 17 14:46

**Armament.cpp**

Page 1/1

```
1  #include "Armament.h"
```

jun 27, 17 14:46

## Table of Content

Page 1/3

Table of Contents					
1	1 Thread.h.....	sheets	1 to 1	( 1) pages	1- 1 21 lines
2	2 Thread.cpp.....	sheets	1 to 1	( 1) pages	2- 2 10 lines
3	3 test_client.cpp.....	sheets	2 to 2	( 1) pages	3- 3 21 lines
4	4 socket.h.....	sheets	2 to 2	( 1) pages	4- 4 47 lines
5	5 socketError.h.....	sheets	3 to 3	( 1) pages	5- 5 28 lines
6	6 socketError.cpp.....	sheets	3 to 3	( 1) pages	6- 6 14 lines
7	7 socket.cpp.....	sheets	4 to 4	( 1) pages	7- 8 125 lines
8	8 messenger.h.....	sheets	5 to 5	( 1) pages	9- 9 48 lines
9	9 messenger.cpp.....	sheets	5 to 5	( 1) pages	10- 10 47 lines
10	10 clientReceiverTest.h	sheets	6 to 6	( 1) pages	11- 11 26 lines
11	11 clientReceiverTest.cpp	sheets	6 to 6	( 1) pages	12- 12 19 lines
12	12 weapon.h.....	sheets	7 to 7	( 1) pages	13- 13 43 lines
13	13 weapon.cpp.....	sheets	7 to 7	( 1) pages	14- 14 43 lines
14	14 unitMold.h.....	sheets	8 to 8	( 1) pages	15- 15 50 lines
15	15 unitMold.cpp.....	sheets	8 to 8	( 1) pages	16- 16 64 lines
16	16 unit.h.....	sheets	9 to 9	( 1) pages	17- 18 92 lines
17	17 unit.cpp.....	sheets	10 to 12	( 3) pages	19- 23 307 lines
18	18 territory.h.....	sheets	12 to 12	( 1) pages	24- 24 43 lines
19	19 territory.cpp.....	sheets	13 to 13	( 1) pages	25- 25 63 lines
20	20 Terrain.h.....	sheets	13 to 13	( 1) pages	26- 26 28 lines
21	21 Terrain.cpp.....	sheets	14 to 14	( 1) pages	27- 27 22 lines
22	22 team.h.....	sheets	14 to 14	( 1) pages	28- 28 26 lines
23	23 team.cpp.....	sheets	15 to 15	( 1) pages	29- 29 27 lines
24	24 teamable.h.....	sheets	15 to 15	( 1) pages	30- 30 42 lines
25	25 teamable.cpp.....	sheets	16 to 16	( 1) pages	31- 31 40 lines
26	26 size.h.....	sheets	16 to 16	( 1) pages	32- 32 60 lines
27	27 size.cpp.....	sheets	17 to 17	( 1) pages	33- 34 85 lines
28	28 server.h.....	sheets	18 to 18	( 1) pages	35- 35 36 lines
29	29 server.cpp.....	sheets	18 to 18	( 1) pages	36- 36 42 lines
30	30 position.h.....	sheets	19 to 19	( 1) pages	37- 37 27 lines
31	31 position.cpp.....	sheets	19 to 19	( 1) pages	38- 38 21 lines
32	32 playerInfo.h.....	sheets	20 to 20	( 1) pages	39- 39 57 lines
33	33 playerInfo.cpp.....	sheets	20 to 21	( 2) pages	40- 41 85 lines
34	34 player.h.....	sheets	21 to 22	( 2) pages	42- 43 68 lines
35	35 player.cpp.....	sheets	22 to 23	( 2) pages	44- 46 149 lines
36	36 Occupant.h.....	sheets	24 to 24	( 1) pages	47- 47 46 lines
37	37 Occupant.cpp.....	sheets	24 to 24	( 1) pages	48- 48 55 lines
38	38 node.h.....	sheets	25 to 25	( 1) pages	49- 49 52 lines
39	39 node.cpp.....	sheets	25 to 25	( 1) pages	50- 50 65 lines
40	40 menu.h.....	sheets	26 to 26	( 1) pages	51- 51 45 lines
41	41 menu.cpp.....	sheets	26 to 27	( 2) pages	52- 54 135 lines
42	42 MapLoader.h.....	sheets	28 to 28	( 1) pages	55- 55 67 lines
43	43 MapLoader.cpp.....	sheets	28 to 30	( 3) pages	56- 59 245 lines
44	44 map.h.....	sheets	30 to 31	( 2) pages	60- 61 86 lines
45	45 map.cpp.....	sheets	31 to 33	( 3) pages	62- 65 200 lines
46	46 main.cpp.....	sheets	33 to 33	( 1) pages	66- 66 63 lines
47	47 lobby.h.....	sheets	34 to 34	( 1) pages	67- 67 57 lines
48	48 lobby.cpp.....	sheets	34 to 35	( 2) pages	68- 70 186 lines
49	49 game.h.....	sheets	36 to 36	( 1) pages	71- 72 73 lines
50	50 game.cpp.....	sheets	37 to 38	( 2) pages	73- 76 211 lines
51	51 factory.h.....	sheets	39 to 39	( 1) pages	77- 77 58 lines
52	52 factory.cpp.....	sheets	39 to 40	( 2) pages	78- 79 113 lines
53	53 controlUnit.h.....	sheets	40 to 41	( 2) pages	80- 81 115 lines
54	54 controlUnit.cpp.....	sheets	41 to 45	( 5) pages	82- 90 530 lines
55	55 compass.h.....	sheets	46 to 47	( 2) pages	91- 93 134 lines
56	56 compass.cpp.....	sheets	47 to 52	( 6) pages	94-104 662 lines
57	57 commandMonitor.h.....	sheets	53 to 53	( 1) pages	105-105 27 lines
58	58 commandMonitor.cpp..	sheets	53 to 53	( 1) pages	106-106 23 lines
59	59 command.h.....	sheets	54 to 54	( 1) pages	107-107 31 lines
60	60 command.cpp.....	sheets	54 to 54	( 1) pages	108-108 66 lines
61	61 cell.h.....	sheets	55 to 55	( 1) pages	109-109 42 lines
62	62 cell.cpp.....	sheets	55 to 55	( 1) pages	110-110 37 lines
63	63 bullet.h.....	sheets	56 to 56	( 1) pages	111-111 67 lines
64	64 bullet.cpp.....	sheets	56 to 57	( 2) pages	112-114 141 lines
65	65 Random.h.....	sheets	58 to 58	( 1) pages	115-115 18 lines

jun 27, 17 14:46

## Table of Content

Page 2/3

66	66 Random.cpp.....	sheets	58 to 58	( 1) pages	116-116 10 lines
67	67 MapGenerator.h.....	sheets	59 to 59	( 1) pages	117-118 73 lines
68	68 MapGenerator.cpp....	sheets	60 to 62	( 3) pages	119-123 303 lines
69	69 main.cpp.....	sheets	62 to 62	( 1) pages	124-124 31 lines
70	70 Thread.h.....	sheets	63 to 63	( 1) pages	125-125 21 lines
71	71 Thread.cpp.....	sheets	63 to 63	( 1) pages	126-126 10 lines
72	72 split.h.....	sheets	64 to 64	( 1) pages	127-127 18 lines
73	73 split.cpp.....	sheets	64 to 64	( 1) pages	128-128 16 lines
74	74 socket.h.....	sheets	65 to 65	( 1) pages	129-129 47 lines
75	75 socketError.h.....	sheets	65 to 65	( 1) pages	130-130 28 lines
76	76 socketError.cpp.....	sheets	66 to 66	( 1) pages	131-131 14 lines
77	77 socket.cpp.....	sheets	66 to 67	( 2) pages	132-133 129 lines
78	78 messenger.h.....	sheets	67 to 67	( 1) pages	134-134 49 lines
79	79 messenger.cpp.....	sheets	68 to 68	( 1) pages	135-135 51 lines
80	80 Lock.h.....	sheets	68 to 68	( 1) pages	136-136 19 lines
81	81 Lock.cpp.....	sheets	69 to 69	( 1) pages	137-137 10 lines
82	82 ResultWindow.h.....	sheets	69 to 69	( 1) pages	138-138 35 lines
83	83 ResultWindow.cpp.....	sheets	70 to 70	( 1) pages	139-139 47 lines
84	84 MenuWindow.h.....	sheets	70 to 70	( 1) pages	140-140 40 lines
85	85 MenuWindow.cpp.....	sheets	71 to 71	( 1) pages	141-141 62 lines
86	86 LobbyWindow.h.....	sheets	71 to 71	( 1) pages	142-142 48 lines
87	87 LobbyWindow.cpp.....	sheets	72 to 72	( 1) pages	143-144 82 lines
88	88 InitialWindow.h.....	sheets	73 to 73	( 1) pages	145-145 36 lines
89	89 InitialWindow.cpp...	sheets	73 to 73	( 1) pages	146-146 58 lines
90	90 GameWindow.h.....	sheets	74 to 74	( 1) pages	147-148 90 lines
91	91 GameWindow.cpp.....	sheets	75 to 76	( 2) pages	149-152 259 lines
92	92 UnitsMonitor.h.....	sheets	77 to 77	( 1) pages	153-153 43 lines
93	93 UnitsMonitor.cpp.....	sheets	77 to 78	( 2) pages	154-155 117 lines
94	94 Unit.h.....	sheets	78 to 79	( 2) pages	156-157 110 lines
95	95 Unit.cpp.....	sheets	79 to 80	( 2) pages	158-160 139 lines
96	96 ServerMessenger.h...	sheets	81 to 81	( 1) pages	161-161 28 lines
97	97 ServerMessenger.cpp..	sheets	81 to 81	( 1) pages	162-162 28 lines
98	98 UnitPanel.h.....	sheets	82 to 82	( 1) pages	163-163 38 lines
99	99 UnitPanel.cpp.....	sheets	82 to 83	( 2) pages	164-165 69 lines
100	100 BuildingPanel.h.....	sheets	83 to 84	( 2) pages	166-167 68 lines
101	101 BuildingPanel.cpp...	sheets	84 to 85	( 2) pages	168-169 101 lines
102	102 Nature.h.....	sheets	85 to 85	( 1) pages	170-170 24 lines
103	103 Nature.cpp.....	sheets	86 to 86	( 1) pages	171-171 15 lines
104	104 MapMonitor.h.....	sheets	86 to 87	( 2) pages	172-173 68 lines
105	105 MapMonitor.cpp.....	sheets	87 to 88	( 2) pages	174-175 129 lines
106	106 Map.h.....	sheets	88 to 88	( 1) pages	176-176 57 lines
107	107 Map.cpp.....	sheets	89 to 89	( 1) pages	177-178 68 lines
108	108 main.cpp.....	sheets	90 to 90	( 1) pages	179-179 48 lines
109	109 Game.h.....	sheets	90 to 90	( 1) pages	180-180 44 lines
110	110 Game.cpp.....	sheets	91 to 91	( 1) pages	181-182 85 lines
111	111 GameBuilder.h.....	sheets	92 to 92	( 1) pages	183-183 43 lines
112	112 GameBuilder.cpp.....	sheets	92 to 93	( 2) pages	184-185 89 lines
113	113 GameArea.h.....	sheets	93 to 96	( 4) pages	186-191 345 lines
114	114 GameArea.cpp.....	sheets	96 to 101	( 6) pages	192-202 679 lines
115	115 UnitsEnum.h.....	sheets	102 to 102	( 1) pages	203-203 14 lines
116	116 TeamEnum.h.....	sheets	102 to 102	( 1) pages	204-204 9 lines
117	117 RotationsEnum.h.....	sheets	103 to 103	( 1) pages	205-205 20 lines
118	118 NatureEnum.h.....	sheets	103 to 103	( 1) pages	206-206 9 lines
119	119 BuildingsEnum.h.....	sheets	104 to 104	( 1) pages	207-207 14 lines
120	120 ActionsEnum.h.....	sheets	104 to 104	( 1) pages	208-208 10 lines
121	121 Counter.h.....	sheets	105 to 105	( 1) pages	209-209 18 lines
122	122 Counter.cpp.....	sheets	105 to 105	( 1) pages	210-210 14 lines
123	123 Winner.h.....	sheets	106 to 106	( 1) pages	211-211 21 lines
124	124 Winner.cpp.....	sheets	106 to 106	( 1) pages	212-212 11 lines
125	125 UpdateUnit.h.....	sheets	107 to 107	( 1) pages	213-213 20 lines
126	126 UpdateUnit.cpp.....	sheets	107 to 107	( 1) pages	214-214 30 lines
127	127 UpdateTerritory.h...	sheets	108 to 108	( 1) pages	215-215 18 lines
128	128 UpdateTerritory.cpp..	sheets	108 to 108	( 1) pages	216-216 33 lines
129	129 UpdatePosition.h...	sheets	109 to 109	( 1) pages	217-217 20 lines
130	130 UpdatePosition.cpp...	sheets	109 to 109	( 1) pages	218-218 20 lines
131	131 UpdateFactory.h.....	sheets	110 to 110	( 1) pages	219-219 20 lines

jun 27, 17 14:46		Table of Content		Page 3/3	
133	132	<i>UpdateFactory.cpp...</i>	sheets 110 to 110 ( 1) pages 220-220	26	lines
134	133	<i>StartGame.h.....</i>	sheets 111 to 111 ( 1) pages 221-221	24	lines
135	134	<i>StartGame.cpp.....</i>	sheets 111 to 111 ( 1) pages 222-222	15	lines
136	135	<i>RemoveUnit.h.....</i>	sheets 112 to 112 ( 1) pages 223-223	19	lines
137	136	<i>RemoveUnit.cpp.....</i>	sheets 112 to 112 ( 1) pages 224-224	11	lines
138	137	<i>PlayerNames.h.....</i>	sheets 113 to 113 ( 1) pages 225-225	18	lines
139	138	<i>PlayerNames.cpp.....</i>	sheets 113 to 113 ( 1) pages 226-226	12	lines
140	139	<i>MapsInfo.h.....</i>	sheets 114 to 114 ( 1) pages 227-227	18	lines
141	140	<i>MapsInfo.cpp.....</i>	sheets 114 to 114 ( 1) pages 228-228	13	lines
142	141	<i>Loser.h.....</i>	sheets 115 to 115 ( 1) pages 229-229	18	lines
143	142	<i>Loser.cpp.....</i>	sheets 115 to 115 ( 1) pages 230-230	11	lines
144	143	<i>LobbyInfo.h.....</i>	sheets 116 to 116 ( 1) pages 231-231	18	lines
145	144	<i>LobbyInfo.cpp.....</i>	sheets 116 to 116 ( 1) pages 232-232	9	lines
146	145	<i>LoadMap.h.....</i>	sheets 117 to 117 ( 1) pages 233-233	26	lines
147	146	<i>LoadMap.cpp.....</i>	sheets 117 to 117 ( 1) pages 234-234	57	lines
148	147	<i>JoinLobby.h.....</i>	sheets 118 to 118 ( 1) pages 235-235	20	lines
149	148	<i>JoinLobby.cpp.....</i>	sheets 118 to 118 ( 1) pages 236-236	23	lines
150	149	<i>FactoryStats.h.....</i>	sheets 119 to 119 ( 1) pages 237-237	19	lines
151	150	<i>FactoryStats.cpp.....</i>	sheets 119 to 119 ( 1) pages 238-238	22	lines
152	151	<i>FactoryNextUnit.h...</i>	sheets 120 to 120 ( 1) pages 239-239	18	lines
153	152	<i>FactoryNextUnit.cpp.</i>	sheets 120 to 120 ( 1) pages 240-240	14	lines
154	153	<i>Command.h.....</i>	sheets 121 to 121 ( 1) pages 241-241	13	lines
155	154	<i>AddUnit.h.....</i>	sheets 121 to 121 ( 1) pages 242-242	22	lines
156	155	<i>AddUnit.cpp.....</i>	sheets 122 to 122 ( 1) pages 243-243	55	lines
157	156	<i>AddNature.h.....</i>	sheets 122 to 122 ( 1) pages 244-244	18	lines
158	157	<i>AddNature.cpp.....</i>	sheets 123 to 123 ( 1) pages 245-245	30	lines
159	158	<i>AddBuilding.h.....</i>	sheets 123 to 123 ( 1) pages 246-246	21	lines
160	159	<i>AddBuilding.cpp.....</i>	sheets 124 to 124 ( 1) pages 247-247	39	lines
161	160	<i>ClientThread.h.....</i>	sheets 124 to 124 ( 1) pages 248-248	54	lines
162	161	<i>ClientThread.cpp....</i>	sheets 125 to 125 ( 1) pages 249-250	103	lines
163	162	<i>Cell.h.....</i>	sheets 126 to 126 ( 1) pages 251-251	22	lines
164	163	<i>Cell.cpp.....</i>	sheets 126 to 126 ( 1) pages 252-252	16	lines
165	164	<i>Camera.h.....</i>	sheets 127 to 127 ( 1) pages 253-253	54	lines
166	165	<i>Camera.cpp.....</i>	sheets 127 to 128 ( 2) pages 254-255	82	lines
167	166	<i>BuildingsMonitor.h..</i>	sheets 128 to 128 ( 1) pages 256-256	39	lines
168	167	<i>BuildingsMonitor.cpp</i>	sheets 129 to 129 ( 1) pages 257-258	87	lines
169	168	<i>Building.h.....</i>	sheets 130 to 130 ( 1) pages 259-260	72	lines
170	169	<i>Building.cpp.....</i>	sheets 131 to 131 ( 1) pages 261-262	121	lines
171	170	<i>Armament.h.....</i>	sheets 132 to 132 ( 1) pages 263-263	23	lines
172	171	<i>Armament.cpp.....</i>	sheets 132 to 132 ( 1) pages 264-264	2	lines