Maliputy Technical Report

Robin v. Grinsven,

NHTV/ IADE/ Europiade.

Contents

Abstract.....5

Maliputy Technical Report.....6

Game Software requirments.....6

Functional requirements.....6

Gameplay.....6

Server Client.....6

Non Functional requirements.....6

Software.....6

Network.....6

Performance.....6

Blueprint functions.....6

Interaction.....6

Game Software design.....6

Software architecture.....7

Coding Style.....7

Blueprinting style.....7

Systems.....7

RPS system.....7

MatchMaking.....7

Team creation....7

Server handeling....7

Game software programming....7

Blueprints....7

Machine....7

Communication (from item till machine)....7

Interaction....7

HUD....7

C++ Blueprints....7

RPS system....7

Peer to Peer messaging....7

C++ Server....7

Data structure....7

Match making....7

Threads....7

MaliPuty Technical report

Abstract

This document covers the programming side of the project. How design is implemented. And how the code works.

Maliputy Technical Report

Maliputy Is a student project that gives the first experience to coders to make a multiplayer project. Where there is a server client structure for match making and a peer to peer connection for in the game time.

Game Software Requirements

Functional requirement

Gameplay

Player need to be able to walk across the map. Collect items and put them in a machine.

Falling of the map makes the players respawn at Spawn point.

The items beat each other in a rock paper scissor.

The level can have many traps. These traps help to make the game more enticing and make the player think more.

Feedback

Good feedback is important for this game. It is the deciding factor of what the player needs to think about what they need to guess and wat is a given.

In the game you see the machine state and the countering items.

Server Client

For this project we needed to make a client server matchmaking this should define who is going to play with each other. This is defined by xp distance. Once found clients connect to each other by the message send by the server.

Peer to Peer

In the game we build a peer to peer system. Where we send least amount of data. We send actions to the players so cheating is hard.

Non Functional requirements

Software

All the main software used for the project development. With explanation what role they played.

Unreal

This is the game engine. All basics of a game are coded here. The programmer works with it's API. To develop games. It provided the framework for the game.

VMware

A Virtual machine simulator so I can have Linux on top of my Windows. This way I can code on Linux for the server while I run an Unreal instance for the client. On 1 machine.

Arch-Linux:

I want to have the server running on Linux. Because it is more tested on being a server. Also arch Linux is the distrobution I use because it is not a desktop distrobution but a server distrobution.

Emacs

Used to code on the server side.

Visual studio

Used to code in the engine.

Network

For the game we need a functional network over the internet. A server needs to be running to match making. But then the players should be working by themselves. After this they should report to the server their status.

Performance

The performance of the game has many aspects to it. First the network is the goal of the semester. So the goal is to try to get least amount of traffic over the network and efficiently. We

try this by thinking of every concept in the game look what is the minimum amount of data necessary. Next to that when something needs to happen after a time period. We directly send the message when the timer needs to start. Not when the action needs to happen so latency has least amount of impact.

Other performance issues are graphical and collision. We take a minimalistic approach to this. Minimal graphics and use

For code we do the not supported algorithms in c++ like the rps system. If we would be doing it in blueprints we are less in control of the performance compared to c++. however when advanced algorithms are build in the blueprint system we use the blueprint version. This saves hassle and we expect the coders of Epic games to know their own engine better.

Blueprint functions

The considerations when to use blueprints are explained above however we also do this for support. In unreal tutorials the support for blueprints is high, in order to solve our problems faster we use blueprints.

Interaction

In the game players interact with the orbs in the level. To pick up place on machine. This is the main interaction of the players. Next to this they collide with the ground and trees.

Players can show HUD elements with click of buttons. And the HUD responds to the play in the game.

Game Software design

Tells the concept behind general idea of the code written

Software architecture

Coding Style

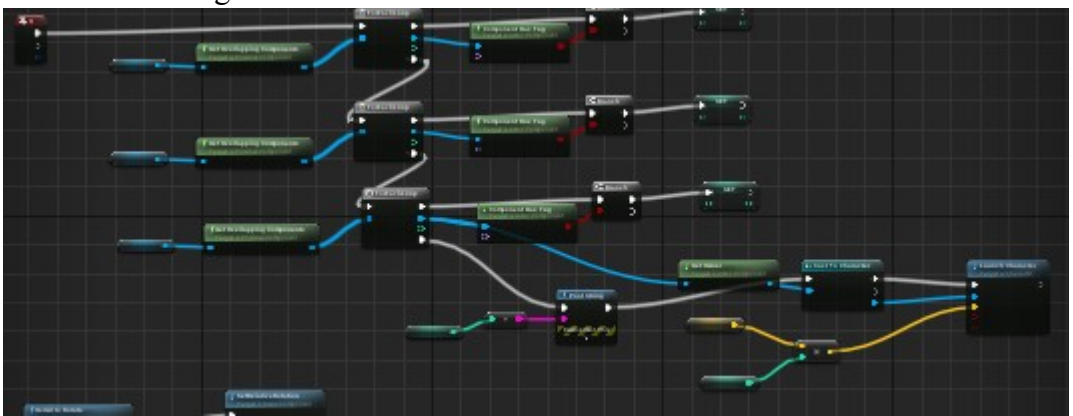
In code I use own imagened “commenting first” technique. When I write a code I first write all the comments that describe what the section is going to do. Then I write the rough code. And refine it based on feedback of the system. 1 line of code can not be responsible for more then 1 action. Example: can only assign to 1 variable. Brackets {} have their own line. 1 line if statement or any function line has the execution code on the next line. Also repeated addition. Like A[N].A + A[N+1].A + A[N+2].C, all get their own line. This way the code does not get clutterd.

```
colC = colB +1;
for(uint8_t rol_C= 0;rol_C<rol_A;rol_C++)
  //calculates the value of combined xp
  closeness = match[0] -> xp +
    match[colB] -> xp +
    match[colC] -> xp -
    totalxp/2;
  closeness = std::abs(closeness);
  //check if the composition is more idea
```

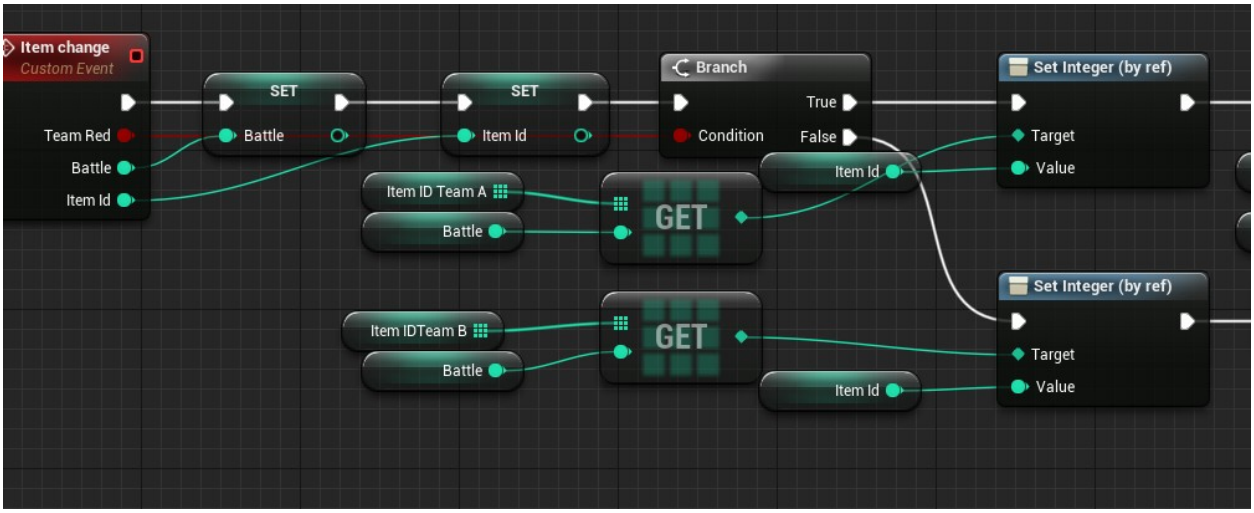
Blueprinting style

Blueprints also follow a style. When code becomes very low level(addition substraction, base manipulators.) we make a C++ code with a blueprint function that describes the higher level function.

When a piece of code needs to be repeated some times we put then in a stacking order like the following



Next to that to keep things clean we use redirect nodes however we do not try to make to many lines cross each other. Next to that we try to keep the execution lines straight. Unless a branch occurs then we do as example below.

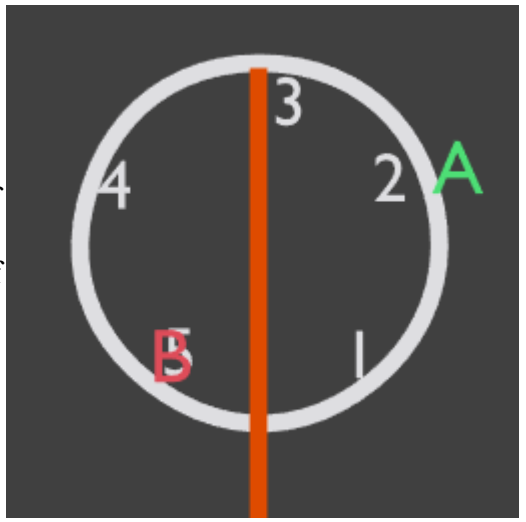


Systems

RPS system

For the RPS system I generated my own algorithm. There might be an equivalent system out there. But this is a simple problem so to train my problem solving skills I took the challenge. Using my own logic.

With the consideration that every object has equal amount of objects it wins from as it losses . We consider for the check to put item A it's value in the middle of the range of possibility's. Check what that velocity is, move the B value to it as well. And map that value back in range of the array with the mod function. Then you can check if the value of B is bigger or smaller then A. and see who wins. Figure: checking if B is behind or before the Red line.



Match Making

If someone comes online and wants to play, we ask for the experience points. We check what group the person belongs in the XP group fashion to and add 1 to the list of players want to play in it. Figure a player adds itself to play status his xp is 490 get added to the place: 400-499

0-99	100-199	200-299	300-399	400-499	500-599	600-699	700-799	800-899	900-999
50	12	2	2	1+1 =2	1	4	2	2	1

Now for instance the player wants to play a game. The algorithm checks if there are enough players in its group. If not the case ti look at its neighbors.

Figure below: checks own group makes 2 players. Player needs 6 players to play so going to look at it's neighbours.

0-99	100-199	200-299	300-399	400-499	500-599	600-699	700-799	800-899	900-999
50	12	2	2	2	1	4	2	2	1

Figure below: checks it first neighbors sees this makes total of 5. still not enoug. So need to look further

0-99	100-199	200-299	300-399	400-499	500-599	600-699	700-799	800-899	900-999
50	12	2	2	2	1	4	2	2	1

Now it looks 1 futher. Sees there are total 11 people online in range that is enough to play the game. He starts looking at who first logged in and once one is fitting the group he adds it to the play list.

0-99	100-199	200-299	300-399	400-499	500-599	600-699	700-799	800-899	900-999
50	12	2	2	2	1	4	2	2	1

He starts at the first logged in. then moves allong the list of connected. That order is determined by FIFO. When he finds a players that fits the range he add it to the list. Until they are with 6 players.

Team creation

When team creation it is ideal to have both teams equal amount of experience points. We do this by checking every option of team combination. However I am not going to look at whole team composition. I just do 1 team starting with 1 standard person this person is fixed. This person get checked with every other players. And find out if the total experience of the team is closest to the middle. The closer to the middle of the total experience of the matched players. The more ideal the team compositions are. The server checks every option once by setting player 1 stuck then checks it together with player 2 and every other player. Then checks with player 3 in slot and checks 4-6. example:

1 = 432 2 = 324 3 = 545 4 = 406 5 = 564 6 = 468

ideal = 1369,5

1	2	3	1301	-68,5	1301
1	2	4	1162	-207,5	1301
1	2	5	1320	-49,5	1320
1	2	6	1224	-145,5	1320
1	3	4	1383	13,5	1383
1	3	5	1541	171,5	1383
1	3	6	1445	75,5	1383
1	4	5	1402	32,5	1383
1	4	5	1306	-63,5	1383
1	5	6	1464	94,5	1383

As the other team is the mirror this should result in testing every option. Now you only replace the teammember in mind with the ideal team member if the next option is better. Ideal here would be 1 with 3 and 4. this gives the other team 1356. very close to eachother.

Server handling

Server handels a fifo system when one logs in it gets on the end of the list. And when match making it checks the first in the list if it wants a match.

Game software programming

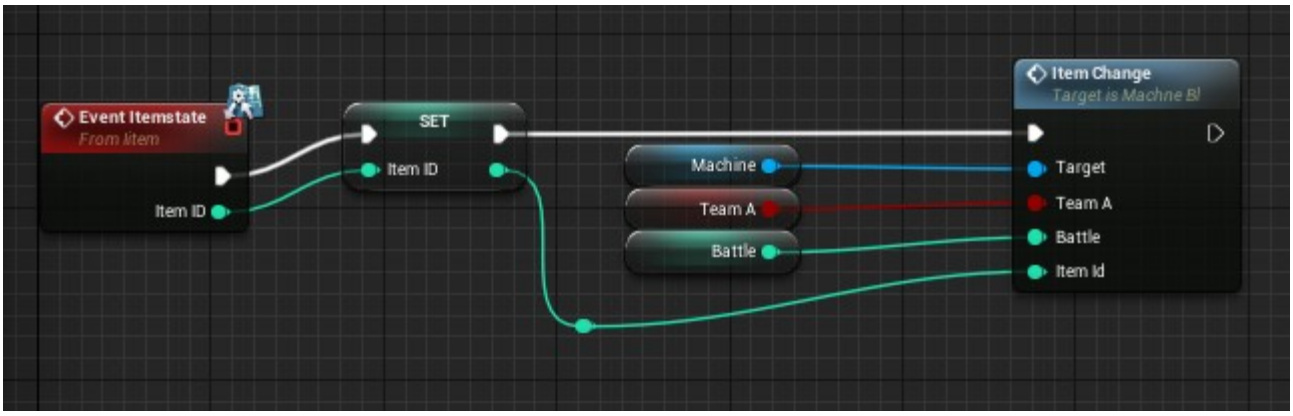
Shows the code and then explain what the code is responsible for.

Blueprints

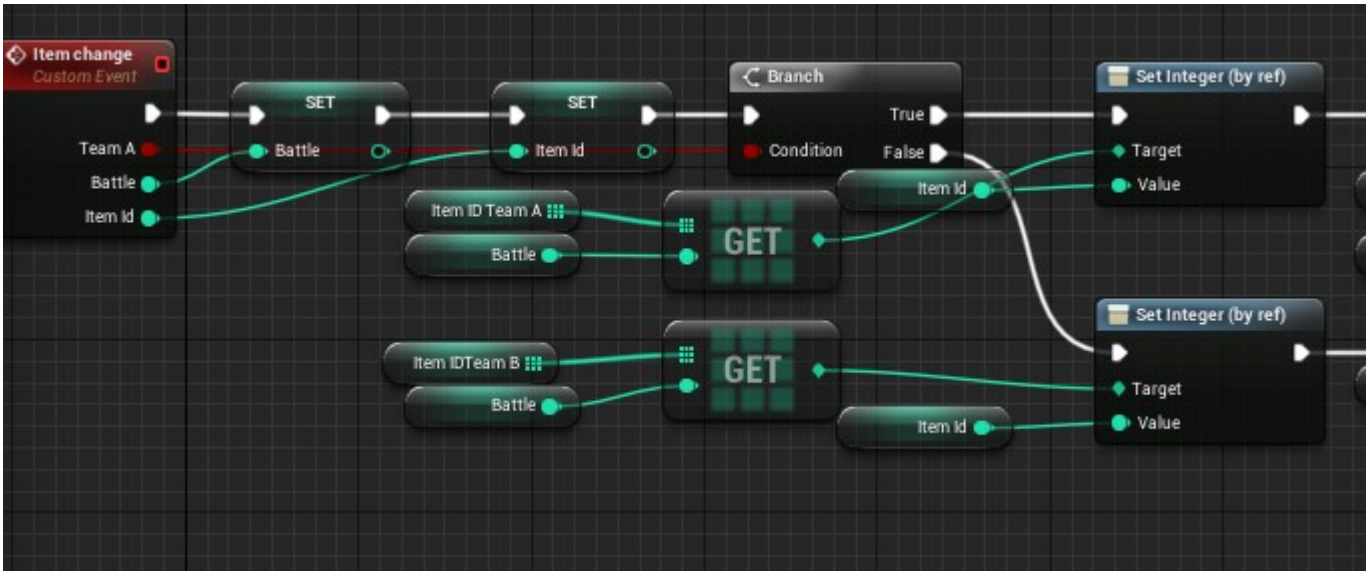
Machine

The machine uses parent and child blueprints so I do not need to create 6 slots.

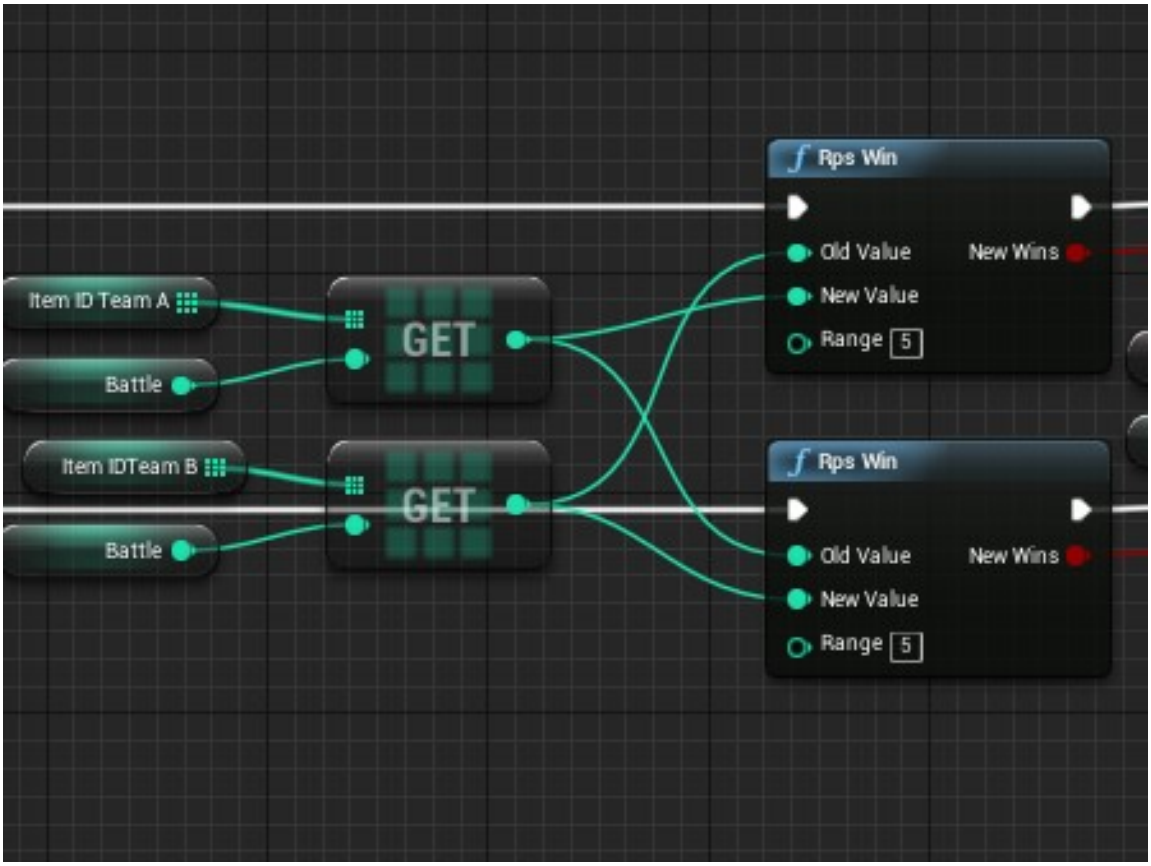
The client just is a collision detector when a item is dropt it sends a signal to the main machine.



When it is send. The main machine sets the new value of the object in the array



then checks with the posing object. You see the c++ code how it get's evaluated.

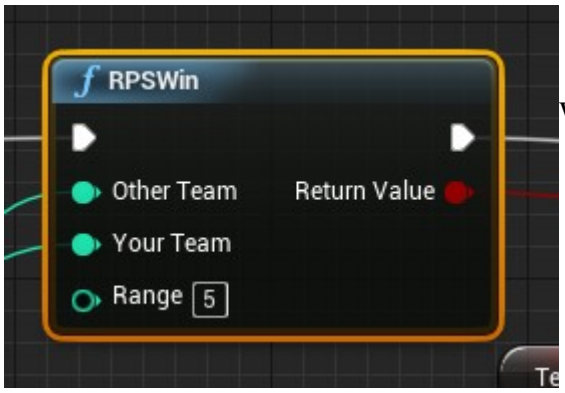


Communication (from item till machine)

We use an interface to comminucate between the needed objects. When a class inheriterds an interface the other knows. And is able to send the int value to the other object, this is used to move the items around the area.

C++ Blueprints

RPS system



We made a node of this algorithm

for it to be calleable it need to inherit the parent class “UblueprintFunctionLibrary”.

The function then expected needs to be a static and a UFUNCTION. BlueprintCallable

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"
#include "Kismet/BlueprintFunctionLibrary.h"
#include "RPS.generated.h"

UCLASS()
class BATTERYCOLLECTOR_API URPS : public UBlueprintFunctionLibrary
{
    GENERATED_BODY()

    UFUNCTION(BlueprintCallable, Category = "GameMech")
        static bool RPSWin(int32 OtherTeam, int32 YourTeam, int32 Range);
};
```

then in the function we can place the code. We first do a bitswift for a division. It has a few less instructions then division by 2 in assembly. Then make sure the data is in range.

After that it looks how far 1 value is from the middle. And moves the other value. And checks if the other is more or less in the middle value. We need to do this since we do not know what the range would be of items. This makes the code flexible for future use.

```
bool URPS::RPSWin(int32 OtherTeam, int32 YourTeam, int32 Range)
{
    int32 Middle = Range >> 1;//find the middle
    YourTeam %= Range;//put your data in range
    OtherTeam %= Range;//put other team data in range
    int32 swift = Middle - OtherTeam;//find the velocity
    YourTeam = (YourTeam + swift)%Range;
    return(YourTeam < Middle);
}
```

C++ Server

Data structure

We make a list of clients. This class is able to modify the list. We start at the beginning of the list when looking for a match. So we start from beginning to end. This holds data of the global stats

```
class clientList{
private:
  client *end = NULL;
  client *start = NULL;
  unsigned int median[median_scale] = {0};// a list of median players online based on xp
level
public:
  unsigned long int players_online =0;
  //basic list functions
  client *getEnd();
  client push(struct sockaddr_in ip, char *name,unsigned int xper);
  client push(client *player);//add new connected client
  client pop(client *player);// deletes client from list and returns it
  //matchmaking functions
  client *pick();//ret urns client.
};
```

The client holds it's own ip adress It socket address to send it a message in any time. Name xp for match making and identity. Hold the next pointer. So it goes trough the list. The team it is pick towards. And a status is it ready to play

```
//state of the player
enum statuses{
  Init,
  Idle,
  Lobby,
  Launching
};
//data of client, to find and create matches.
struct client{
  sockaddr_in ip_address;
  void *sock;
  unsigned int xp;
  std::string name;
  client *next = NULL;// next client in list
  unsigned int Team; // assigend to team in match
  statuses status = Init;
};
```

Match making

Init list:

```
client *clientList::pick(){
  client **match =(client**) std::malloc(sizeof(struct client) * Players);
  client *proposal = start -> next;//player currently checked if wanted to have
  match[0] = start;
  for(int i =1; i < Players; i++)
    match[i] = new client;
```

```
//define range to look at

unsigned int firstP_median = match[0] -> xp / median_sizestep;

unsigned int MaxDistance = 0;
unsigned int players_inrange = median[firstP_median];//ammount of players in range
//checks if there are enough players in range
while(players_inrange < Players){
  MaxDistance +=1;
  //catches if there are not enough players online
  if(firstP_median - MaxDistance <= 0 &&  firstP_median + MaxDistance >= median_scale){
    std::cout << "not enough players." << std::endl;
    players_inrange += Players;
  }
  // if one still in range
  if(firstP_median-MaxDistance >= 0){
    players_inrange += median[(firstP_median-MaxDistance)];
  }
  if( firstP_median + MaxDistance <= median_scale){
    players_inrange += median[(firstP_median+MaxDistance)];
  }
}
//find close enough xp players based
 while(match[Players-1] -> next != NULL){
  unsigned int proposalmedian = proposal -> xp / median_sizestep;
  int playernum = 1;
  //add player if in right group
  int distancetoFirst = proposalmedian - firstP_median;
  if((unsigned int)std::abs(distancetoFirst) <= MaxDistance  ){
    match[playernum] = proposal;
    playernum++;
  }
  proposal = proposal -> next;
}
```

then to match the teams it gives the players the value of what team they are in:

```
  ,
  //find xp balance in teams and assign to teams.
  uint8_t colB = 1;
  uint8_t colC = 2;
  //total value divided by 2. because that is the ideal value.
  int closeness;
  unsigned int totalxp = 0;
  for(int i=0; i< Players;i++ ){
    totalxp += match[i] -> xp;
  }
  //test every combination
  unsigned int old= -1;//this hold the value that is the closest to 0 for finding th idea
l match.
  uint8_t IdealB;
  uint8_t IdealC;
  for(uint8_t rol_A = Players/2+1; rol_A>0; rol_A--){
    colC = colB +1;
    for(uint8_t rol_C= 0;rol_C<rol_A;rol_C++){
      //calculates the value of combined xp
      closeness = match[0] -> xp +
        match[colB] -> xp +
        match[colC] -> xp -
        totalxp/2;
      closeness = std::abs(closeness);
      //check if the composition is more ideal then previews situations
      if((unsigned int)closeness < old){
        IdealB = colB;
        IdealC = colC;
      }
      colC++;
    }
    colB++;
  }
  //assign people to the teams
  match[0] -> Team = 0;
  for(int i = 1; i < Players; i++){
    if(i == IdealB || i == IdealC){
      match[i] -> Team = 0;
    }
    else{
      match[i] ->Team = 1;
    }
  }
  return *match;
}
```

Threads

After there are enough players online server checks for matches. And send them ip addresses that fit the scale.

```
    if(playlist.players_online >= Players)
      {
        std::cout << "wait" << std::endl;
        //make the avarage wait 1 minute for match making
        usleep((useconds_t)(playlist.players_online/Players*5000));
        std::cout << "play" << std::endl;
        client *match = playlist.pick();
        int *Psock, Sock;
        for(int i = 0; i < Players; i++)
          {
            std::cout << "guess_s" << std::endl;
            Psock = (int*)match[i].sock;
            std::cout << "guess_u" << std::endl;
            Sock = *Psock;
            std::cout << "guess_v" << std::endl;
            std::string message = "";
            //constructs a message
            for(int j = 0; j < Players; j++)
              {
                //if(i != j)
                //{

                    std::cout << "guess_a" << std::endl;
                    //add ip to message
                    message.append("Ip: ");
                    message.append(inet_ntoa(match[j].ip_address.sin_addr));
                    //add name to message
                    message.append(" Name: ");
                    message.append(match[j].name);
                    message.append(" | ");
                    //}
              }

            std::cout << "guess_l" << std::endl;
            //puts(message.c_str());
            write(Sock,message.c_str(),strlen(message.c_str()));
            //playlist.pop(&match[i]);
          }
        std::cout << "guess_0" << std::endl;
        for(int i = 0; i < Players; i++)
          playlist.pop(&match[i]);
        std::cout << "guess_1" << std::endl;
        free(match);
      }
    //Now join the thread , so that we dont terminate before the thread
```

Threads

Using threads to keep multiple clients responsive

```
oid *connection_handler(void *Player)

//Get the socket descriptor
client *player = (client*)Player;
int *Psock =(int*)player -> sock;
int sock = *Psock;
int read_size;
char *message, client_message[2000];

//Send some messages to the client
message = (char*) "Greetings! I am your connection handler\n";
write(sock , message ,strlen(message));//.length());
message = (char*)"Now type something and i shall repeat what you type \n";
write(sock , message , strlen(message));
//Receive a message from client
while( (read_size = recv(sock , client_message , 2000 , 0)) > 0 )
  {
    //Send the message back to client
    if(strncmp(client_message,"play",4) ==  0 && strlen(client_message) > 17)
      {
        std::string Smessage;
        message =(char*) ("Wait! \n");
        write(sock, message, strlen(message));
        Smessage.assign(client_message);
        player -> xp =  atoi(Smessage.substr(5,6).c_str());
        player -> name = Smessage.substr(12);
        player -> status = Lobby;
        std::cout << player -> xp <<
          " | " << player -> name
                << std::endl;
      }
    else{

      write(sock , client_message , strlen(client_message));
    }
  }

if(read_size == 0)
  {
    puts("Client disconnected");
    delete(player);
    fflush(stdout);
  }
else if(read_size == -1)
  {
    perror("recv failed");
  }
//Free the socket pointer
free(Psock);

return 0;
```