МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики Кафедра алгоритмических языков

Отчет о выполнении задания практикума

Сетевая игра "Танки"

Баев Тимур, 425 группа

Комиссаров Андрей, 424 группа

Москва, 2018

Оглавление

[**Постановка задачи** 3](#_Toc510727213)

[**Диаграмма основных классов** 4](#_Toc510727214)

[**Спецификации интерфейса** 5](#_Toc510727215)

[**Инструментальные средства** 6](#_Toc510727216)

[**Файловая структура системы** 6](#_Toc510727217)

[**Пользовательский интерфейс** 6](#_Toc510727218)

# 

# **Постановка задачи**

Пользователь запускает программу и выбирает быть сервером или клиентом.  
В первом случае пользователь пишет только порт, во втором- IP и порт.

Как только пара пользователей найдена, начинается раунд.  
Игроки управляют своими танками в двумерном пространстве.  
Цель каждого - набрать побольше баллов за попадания в чужой танк.  
В арсенале каждого есть несколько различных снарядов.

Раунд заканчивается, как только набран максимум баллов.  
Победитель выбирается по количеству баллов.

Далее игрокам предлагается сыграть еще один раунд.

# **Диаграмма основных классов**

# **Спецификации интерфейса**

Текстовая спецификация интерфейса основных классов системы

**class** Basic\_Blow: *# Скелет любого взрыва*

**def** \_\_init\_\_(self,engine,x,y)

**def** get\_name(self) *# return name of this blow*

**def** getXY(self) *# return position for this object*

**def** next(self,\_timer=0) *# move missle for next dx,dy*

**def** done(self) *# return True if method "next" is over*

**def** reroze(self,\_timer=0) *# destroy itself and create extra blow or missle objects in case of need*

**def** draw(self) *# return dict with info how to draw this object*

**class** Basic\_Missile:

**def** \_\_init\_\_(self,engine,x,y,power,angle)

**def** get\_name(self) *# return name of this misslie*

**def** getXY(self) *# return position for this object*

**def** next(self,\_timer=0) *# move missle for next dx,dy*

**def** done(self) *# return True if method "next" is over or False*

**def** reroze(self,\_timer=0) *# destroy itself and create extra blow or missle objects in case of need*

**def** draw(self) *# return dict with info how to draw this object*

**class** Basic\_Tank:

**def** \_\_init\_\_(self,engine,x,y,power=50,angle=None,weapons=[Basic\_Missile,Firecracker\_Missile,Shotgun\_Missile])

**def** same\_press(self,task) *# return weight if same button was pressed several times at once*

**def** get\_current\_weapon\_name(self) *# return current weapon name*

**def** get\_power(self) *# return current tank's power*

**def** near(self,X,Y) *# returns True if current tank is near point(X,Y)*

**def** get\_angle(self) *# return current tank's angle,return new X*

**def** move(self) *# move changes X according to move\_counter*

**def** getXY(self) *# return current position for this obj*

**def** draw(self) *# return dict with info how to draw this object*

**def** promise\_move(self,task) *# save promise to move, task ::= left | right (str)*

**def** change\_angle(self,task) *# changes angle , task ::= add | sub (str)*

**def** change\_power(self,task) *# changes power , task ::= add | sub (str)*

**def** change\_weapon(self,task) *# changes weapon , task ::= next | prev (str)*

**def** fire(self) *# FIRE!!!!!!!*

**class** Connector:

**def** \_\_init\_\_(self,mode,host,port)

**def** connect(self) *# connect to peer*

**def** success(self) *# return True if there are opened socket*

**def** close(self)

**def** \_\_send(self,sock,msg) *# low-level send msg*

**def** \_\_recv(self,sock) *# low-level recv msg*

**def** get\_ping(self,sock) *# recv ping signal,return True if got ping or False*

**def** send\_ping(self,sock) *# just send ping signal*

**def** \_\_answer\_ping(self) *# answer for ping request*

**def** read\_msg(self) *# recv msg from socket,return bytes in case of success or None if socket is closed*

**def** write\_msg(self,msg) *# send msg into socket*

**def** send\_weight(self,weights) *# send weight's to peer*

**def** send\_tanks(self,left\_x,right\_x) *# send tanks' coordinates to peer*

**def** ping(self) *# just ping peer*

**class** Controller:

**def** \_\_init\_\_(self)

**def** run\_command(self,dick)

**class** Engine:

**def** \_\_init\_\_(self,canvas,draw\_landscape=True)

**def** f(self)

**def** \_\_find\_seed(self) *# return weights*

**def** \_\_generate(self,weights) *# return pixels*

**def** \_\_draw\_landscape(self) *# just draw landscape and nothing more*

**def** blow\_landscape(self,X,Y,R) *# u know X, u can get Y, u should create explosion funnel in (X,Y) with radius R*

**def** print\_current(self) *# print on canvas the current score*

**def** print\_end(self) *# print on canvas, that game is over and smbd won or game is over*

**def** check\_game(self) *# return True if any user got score >= max score, if True => game is over*

**def** clean(self,elements=None) *# delete all moveble objects from canvas*

**def** get\_weights(self) *# just return weights and nothing else*

**def** set\_weights(self,weights) *# just set new weights and nothing else*

**def** \_\_draw\_obj(self,obj,X,Y) *# get dict with info how to draw obj and it's current position and draw it*

**def** draw\_landscape(self) *# redraw landscape*

**def** single\_draw(self) *# single draw of all moveble objects*

**def** stop(self) *# set stop flag*

**def** get\_pixel(self,x) *# return landscape Y for current x*

**def** is\_ready(self) *# return True if engine is ready for game*

**def** add\_missile\_or\_blow(self,obj) *# add new missile or blow*

**def** add\_tank(self,tank,pos) *# add's new tank*

**def** place\_tanks(self,x1=None,x2=None) *# place tanks*

**class** MainMenu:

**def** \_\_init\_\_(self):

**def** get\_result(self) *# return the result*

**def** Radiobutton\_1\_click(self,x) *# client radiobutton enabled*

**def** Radiobutton\_2\_click(self,x) *# server radiobutton enabled*

**def** start\_button\_click(self,x) *# start button pressed*

**def** exit\_button\_click(self,x) *# exit button pressed*

**def** run(self) *# run GUI*

**class** Screen: *# Screen with main picturebox*

**def** \_\_init\_\_(self,conn,left=True):

**def** run(self) *# main loop*

**def** place\_tanks(self,x1=None,x2=None) *# place tanks*

**def** fork(self) *# start controller's*

**def** wait(self) *# wait for both threads*

**def** check\_game(self): *# return True if any of threads are dead, if True => game is over*

**def** stop\_game(self,event) *# instructions to stop the game*

**def** draw\_picture(self) *# callback from timer to draw picture*

**def** draw\_landscape(self) *# callback from timer to draw picture*

**class** Socket\_Controller(controller.Controller):

**def** \_\_init\_\_(self,conn,engine,left=True)

**def** stop(self) *# set stop flag*

**def** loop(self) *# main loop # do all that should do*

**class** User\_Controller(controller.Controller):

**def** \_\_init\_\_(self,conn,engine,left=True)

**def** command(self,dic) *# command to run & send to peer*

**def** press(self,event) *# button reactions*

**def** stop(self) *# set stop flag*

**def** loop(self) *# just loop*

# **Инструментальные средства**

Язык программирования: Python

Среда разработки:

Используемые библиотеки: tkinter, math, time, json,socket,binascii,threading,random,numpy, scipy.interpolate,traceback,keyboard

# **Файловая структура системы**

Каждый класс лежит в своем файле

basic\_blow.py

basic\_missile.py

basic\_tank.py

conf.py

connector.py

controller.py

engine.py

firecracker\_blow.py

firecracker\_blow2.py

firecracker\_missile.py

firecracker\_missile2.py

main.py

mainmenu.py

screen.py

shotgun\_blow.py

shotgun\_missile.py

shotgun\_missile2.py

socket\_controller.py

user\_controller.py

# **Пользовательский интерфейс**